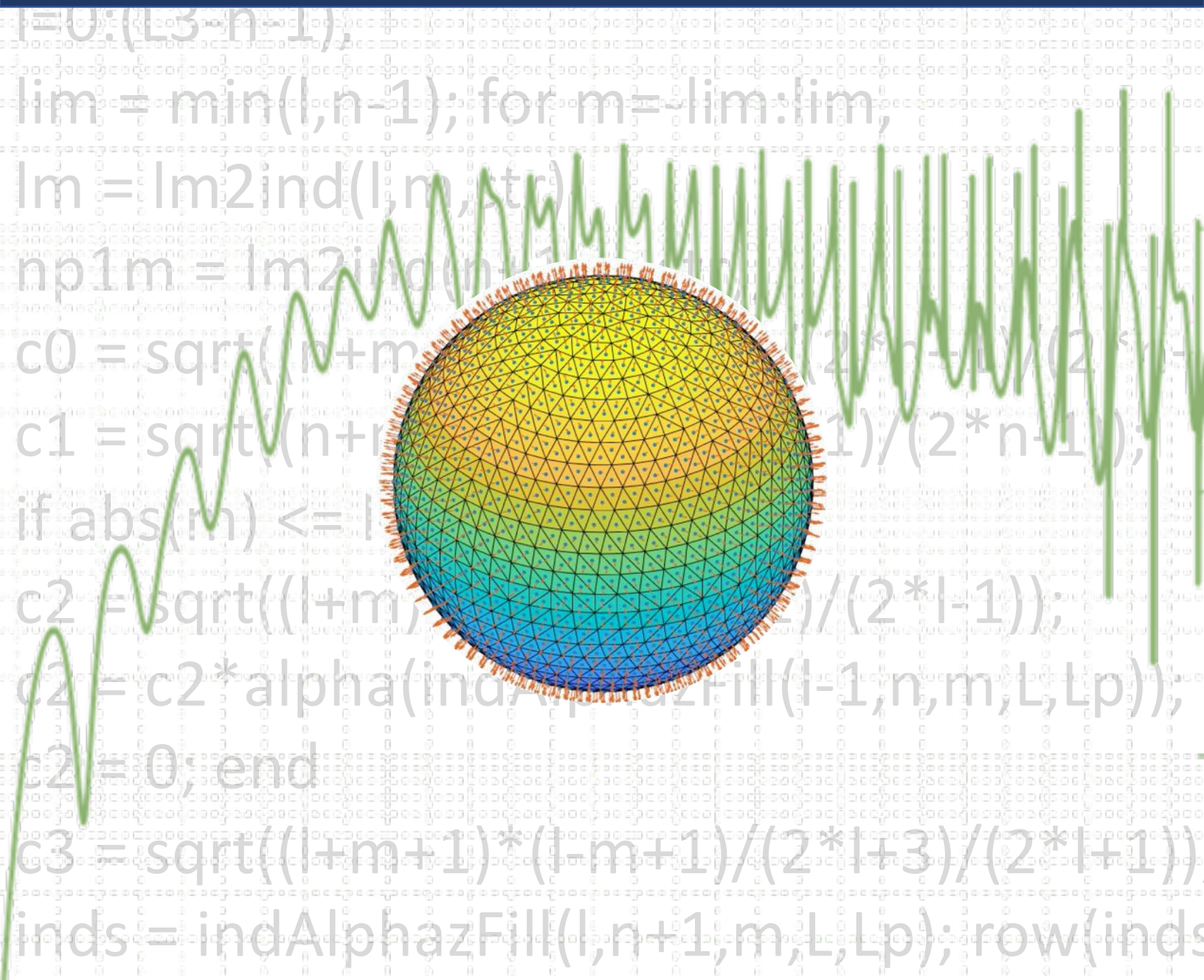
Waveport Scattering Library

Version 1.0

Mark S. Haynes



Waveport Scattering Library

Version 1.0

Mark S. Haynes

September, 2021

Contents

Chapter 1

Introduction

The Waveport Scattering Library is a collection of equations, derivations, and Matlab codes on selected topics in electromagnetic scattering. This library started as a personal collection of notes and codes that became a project to create a consistent code library to which new routines could be added and checked more quickly. It is also a consequence of coding up papers and textbooks over the course of studying and analyzing radar and scattering. We hope this can be used by others to learn the methods, cross-check work, develop better routines, or port to other languages.

Many of the topics follow closely the textbooks of Chew, Tsang, Kong, and Ulaby:

- Green's functions
- Spherical wave functions
- Spherical wave function rotation and translation
- S-matrix and T-matrix
- Kirchhoff approximation and facet scattering
- Fast Multipole Method operators
- Generation of rough surfaces and random objects
- Geometric reflection and refraction
- Coordinate transforms and special functions

Most of the routines are building blocks for analysis rather than heavy-hitting numerical codes. Each routine was either needed at some point, written for curiosity, or thought generally useful. While routines covering many of these topics can be found elsewhere, and done better, some of the most useful algorithms are tricky to code and harder to check. While Matlab is not the programming language of computational electromagnetics, it is quite good for plotting and debugging. Example scripts for running each routine are included in the repository in each topic directory. Input checking is light and not all corner cases have been tested.

The documentation (this document) is written in the style of Numerical Recipes. Equations, derivations, code explanation, and code are combined inline in the text. The idea was to create something that was part quick reference, part code explanation, and part teaching guide. The most complicated algorithms are taken from the literature. Whole derivations are sometimes included, but many are just sketches, and some sections exist for reference without code. Finally, there are many topics that we hope will be included in future releases of the library.

Mark S. Haynes
Jet Propulsion Laboratory, California Institute of Technology
September, 2021

1.1 Link, License, and Citation

The repository for the code and document is:

https://github.com/nasa-jpl/Waveport

The code is released under the Apache 2.0 Open Source license. It is free to run and modify for non-commercial use.

This work can be cited as:

Haynes, M. (2021). *Waveport Scattering Library.* Jet Propulsion Laboratory - California Institute of Technology. https://doi.org/10.48588/JPL.HE9D-BA55

Specific sections and subsections can be cited as, for example:

Haynes, M. (2021). "Spherical Harmonics." *Waveport Scattering Library.* Jet Propulsion Laboratory - California Institute of Technology. https://doi.org/10.48588/JPL.HE9D-BA55

1.2 Acknowledgements

This release would not have been possible without the help of several people. Specifically, Jessie Duan, Ines Fenni, and Richard Chen helped edit and proofread the final draft over several months. The polish is a testament to their skill and expertise in these areas, and their extraordinary attention to detail. Thanks also to Dragana Perkovic-Martin who helped us secure funding. In addition, thanks to the JPL Library staff for help setting up the DOI, and thanks to Diane Fiander and Rebecca Silva for help on the cover art. Finally, thanks to many people who have encouraged me to publish this.

Part of this research was carried out at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration (80NM0018D0004).

1.3 Copyright and Export

1.4 Future Topics

Ideas for future topics include:

- Compendium of examples
- 1D scattering solutions (e.g., recursive multilayer, piece-wise linear)
- 2D scattering solutions (e.g., recursive cylinder solutions, 2D addition theorems)
- Generating time-domain signals from frequency-domain scattering solutions
- Half-space Green's functions in 1D, 2D, and 3D
- Characteristic Basis Function Method
- T-matrix topics: multilayer dielectric sphere, cylinders, cube
- Gaussian tapered incident fields
- Scattering from rough surfaces and rough facets under the Kirchhoff approximation
- Imaging and inverse scattering
- Scattering statistics and sample moments
- Microwave measurement: VNA calibration (1-port and 2-port), RCS of corner reflectors, antenna transmit/receive coefficients.

Chapter 2

Green's Functions

Green's functions are a cornerstone of scalar and vector scattering analysis. Fundamentally, a Green's function is the impulse response of a partial differential equation, in this case, a wave equation with associated boundary conditions. The classic text on dyadic Green's functions for vector electromagnetic waves is [1], along with [2, 3].

Scalar and dyadic Green's functions allow us to cast scattering problems as surface or volume integral equations. These enable solutions of the scattered fields through the use of powerful numerical methods like the Method of Moments and the Conjugate Gradient FFT. In addition, Green's functions help reveal the nature of wave propagation: for example, in 3D free-space, the Green's function shows that the amplitude of waves radiated from a point source decay geometrically with distance and that information is carried undiminished in the phase of the wave.

In this chapter, we list for reference the scalar Green's function in 1D, 2D, and 3D, with far-field expressions for 2D and 3D. We provide routines for the dyadic Green's function for vector electromagnetic waves. We give the volume integral equations (VIE) for scalar and vector waves as well as the expression for the VIE under the far-field Born approximation. The Green's function is singular when evaluated at the origin, however this singularity is integrable. We give the results for the volume-integrated Green's function at singular and non-singular points which are needed when discretizing the VIE. Last, we setup the Method of Moments solution of the VIEs for both scalar and vector cases.

2.1 Scalar Green's Function

The scalar Green's function is the solution to the Helmholtz wave equation for a point source in a homogeneous medium

$$\nabla^2 g(\mathbf{r}, \mathbf{r}') + k^2 g(\mathbf{r}, \mathbf{r}') = -\delta(\mathbf{r} - \mathbf{r}') \tag{2.1}$$

where k is the background wavenumber, $\mathbf{r}$ is the observation point, and $\mathbf{r}'$ is the source point. This wave equation is linear, therefore a scalar field, $\phi(\mathbf{r})$, due to a distributed source, $s(\mathbf{r})$, is given by the volume integral over the Green's function as, [2]

$$\phi(\mathbf{r}) = -\int g(\mathbf{r}, \mathbf{r}') s(\mathbf{r}') dV' \tag{2.2}$$

In short, the Green's function is the impulse response of the linear system that is the wave equation for homogeneous media.

2.1.1 1D Scalar Green's Function

The 1D free-space scalar Green's function is

$$g(x, x') = \frac{i}{2k} e^{ik|x-x'|} \tag{2.3}$$

where x is the observation point, x' is the source point, and k is background wavenumber. This has no far-field approximation because there is no wave spreading in 1D.

2.1.2 2D Scalar Green's Function

The 2D free-space scalar Green's function is

$$g(\boldsymbol{\rho}, \boldsymbol{\rho}') = \frac{i}{4} H_0^{(1)}\left(k|\boldsymbol{\rho} - \boldsymbol{\rho}'|\right) \tag{2.4}$$

where $\boldsymbol{\rho}$ is the observation point, $\boldsymbol{\rho}'$ is the source point, k is the background wavenumber and $H_0^{(1)}$ is the Hankel function, [2]. This is the solution to the Helmholtz wave equation in a homogenous medium due to 2D line source. This has far-field approximation

$$g(\boldsymbol{\rho}, \boldsymbol{\rho}') \approx \sqrt{\frac{1}{8\pi k\rho}} e^{i\pi/4} e^{ik\rho} e^{-ik\hat{\boldsymbol{\rho}}\cdot\boldsymbol{\rho}'} \tag{2.5}$$

This shows that fields decay like $1/\sqrt{\rho}$ in 2D.

2.1.3 3D Scalar Green's Function

The 3D free-space scalar Green's function is

$$g(\mathbf{r}, \mathbf{r}') = \frac{e^{ik|\mathbf{r}-\mathbf{r}'|}}{4\pi|\mathbf{r} - \mathbf{r}'|} \tag{2.6}$$

where k is the background wavenumber, $\mathbf{r}'$ is the source point, $\mathbf{r}$ is the observation point. This is the solution to the Helmholtz wave equation due to a point source in three dimensions. The far-field approximation is

$$g(\mathbf{r}, \mathbf{r}') \approx \frac{e^{ikr}}{4\pi r} e^{-ik\hat{\mathbf{r}}\cdot\mathbf{r}'} \tag{2.7}$$

This shows the classic result that fields decay like $1/r$ in 3D.

2.2 Dyadic Green's Function

The free-space electric field dyadic Green's function satisfies the vector wave equation [2, 1]

$$\nabla \times \nabla \times \overline{\mathbf{G}}(\mathbf{r}, \mathbf{r}') - k^2 \overline{\mathbf{G}}(\mathbf{r}, \mathbf{r}') = \bar{\mathbf{I}}\delta(\mathbf{r} - \mathbf{r}') \tag{2.8}$$

where $\bar{\mathbf{I}}$ is the identity dyad. This is often notated $\overline{\mathbf{G}}_e(\mathbf{r}, \mathbf{r}')$ to distinguish it from the magnetic field dyadic Green's function. The electric field due to a distributed current density $\mathbf{J}(\mathbf{r})$ is given by

$$\mathbf{E}(\mathbf{r}) = i\omega\mu \int \overline{\mathbf{G}}(\mathbf{r}, \mathbf{r}') \cdot \mathbf{J}(\mathbf{r}') dV' \tag{2.9}$$

The dyadic Green's function is given by

$$\overline{\mathbf{G}}(\mathbf{r}, \mathbf{r}') = \left[\bar{\mathbf{I}} + \frac{1}{k^2}\nabla\nabla\right] \frac{e^{ik|\mathbf{r}-\mathbf{r}'|}}{4\pi|\mathbf{r} - \mathbf{r}'|} \tag{2.10}$$

where k is the background wavenumber, $\mathbf{r}'$ is the source point, $\mathbf{r}$ is the observation point, and

$$\nabla = \frac{d}{dx}\hat{x} + \frac{d}{dy}\hat{y} + \frac{d}{dz}\hat{z} \tag{2.11}$$

Writing out (2.10),

$$\overline{\mathbf{G}}(\mathbf{r},\mathbf{r}') = \begin{bmatrix} k^2+\dfrac{\partial^2}{\partial x^2} & \dfrac{\partial^2}{\partial x\partial y} & \dfrac{\partial^2}{\partial x\partial z} \\ \dfrac{\partial^2}{\partial y\partial x} & k^2+\dfrac{\partial^2}{\partial y^2} & \dfrac{\partial^2}{\partial y\partial z} \\ \dfrac{\partial^2}{\partial z\partial x} & \dfrac{\partial^2}{\partial z\partial y} & k^2+\dfrac{\partial^2}{\partial z^2} \end{bmatrix} \frac{e^{ik|\mathbf{r}-\mathbf{r}'|}}{4\pi k^2|\mathbf{r}-\mathbf{r}'|} \tag{2.12}$$

The dyadic Green's function is symmetric with six unique dyadic components. In Cartesian coordinates the components are

$$G_{xx} = (1+c_1(x-x')^2+c_2)c_3 \tag{2.13}$$
$$G_{yy} = (1+c_1(y-y')^2+c_2)c_3 \tag{2.14}$$
$$G_{zz} = (1+c_1(z-z')^2+c_2)c_3 \tag{2.15}$$
$$G_{xy} = c_1(x-x')(y-y')c_3 \tag{2.16}$$
$$G_{xz} = c_1(x-x')(z-z')c_3 \tag{2.17}$$
$$G_{yz} = c_1(y-y')(z-z')c_3 \tag{2.18}$$

where

$$r = |\mathbf{r}-\mathbf{r}'| \tag{2.19}$$
$$c_1 = -\frac{1}{r^2}-\frac{3i}{kr^3}+\frac{3}{k^2r^4} \tag{2.20}$$
$$c_2 = \frac{i}{kr}-\frac{1}{k^2r^2} \tag{2.21}$$
$$c_3 = \frac{e^{ikr}}{4\pi r} \tag{2.22}$$

The far field approximation of the dyadic Green's function is

$$\overline{\mathbf{G}}(\mathbf{r},\mathbf{r}') \approx \left[\overline{\mathbf{I}}-\hat{r}\hat{r}\right]\frac{e^{ikr}}{4\pi r}e^{-ik\hat{\mathbf{r}}\cdot\mathbf{r}'} \tag{2.23}$$

where $\overline{\mathbf{I}}-\hat{r}\hat{r} = \hat{\theta}\hat{\theta}+\hat{\phi}\hat{\phi}$ in spherical coordinates. The curl of the dyadic Green's function is used in surface integral equations and is given by

$$\nabla\times\overline{\mathbf{G}}(\mathbf{r},\mathbf{r}') = \nabla g(\mathbf{r},\mathbf{r}')\times\overline{\mathbf{I}} \tag{2.24}$$

Written out it is

$$\nabla\times\overline{\mathbf{G}}(\mathbf{r},\mathbf{r}') = \begin{bmatrix} 0 & -\dfrac{\partial}{\partial z} & \dfrac{\partial}{\partial y} \\ \dfrac{\partial}{\partial z} & 0 & -\dfrac{\partial}{\partial x} \\ -\dfrac{\partial}{\partial y} & \dfrac{\partial}{\partial x} & 0 \end{bmatrix} \frac{e^{ik|\mathbf{r}-\mathbf{r}'|}}{4\pi|\mathbf{r}-\mathbf{r}'|} \tag{2.25}$$

The unique components, to within a sign, are

$$\left[\nabla\times\overline{\mathbf{G}}(\mathbf{r},\mathbf{r}')\right]_{xy} = -(z-z')f(r) \tag{2.26}$$
$$\left[\nabla\times\overline{\mathbf{G}}(\mathbf{r},\mathbf{r}')\right]_{xz} = (y-y')f(r) \tag{2.27}$$
$$\left[\nabla\times\overline{\mathbf{G}}(\mathbf{r},\mathbf{r}')\right]_{yz} = -(x-x')f(r) \tag{2.28}$$
$$f(r) = (ikr-1)\frac{e^{ikr}}{4\pi r^3} \tag{2.29}$$

A useful property is

$$\nabla\times\overline{\mathbf{G}}(\mathbf{r},\mathbf{r}') = -\nabla'\times\overline{\mathbf{G}}(\mathbf{r},\mathbf{r}') \tag{2.30}$$

The routine `dyadicGreens` takes as input the wavenumber k, and components of the difference vector $\mathbf{r}-\mathbf{r}'$ in Cartesian coordinates and returns the six unique matrix entries of the dyadic Green's function. The routine `curlDyadicGreens` takes the same inputs and returns the six matrix entries of the curl of the dyadic Green's function. These are straight computations, no provisions are made for the singularity.

```
function [Gxx Gyy Gzz Gxy Gxz Gyz] = dyadicGreens(k,X,Y,Z)
% Electromagnetic free space dyadic Green's function in
% Cartesian coordinates. No provisions for r=0.
%
% k:        background wavenumber
% X:        (x-x')
% Y:        (y-y')
% Z:        (z-z')
%       X, Y, and Z are the same size
%
% [Gxx Gyy Gzz Gxy Gxz Gyz]: Unique matrix entries of G(r,r')

r = sqrt(X.^2 + Y.^2 + Z.^2);
c1 = -1./(r.^2)-3*1i./k./(r.^3)+3./(k.^2)./(r.^4);
c2 = 1i./k./r-1./(k.^2)./(r.^2);
c3 = exp(1i*k.*r)./(4*pi*r);
Gxx = (1+c1.*X.^2+c2).*c3;
Gyy = (1+c1.*Y.^2+c2).*c3;
Gzz = (1+c1.*Z.^2+c2).*c3;
Gxy = c1.*X.*Y.*c3;
Gxz = c1.*X.*Z.*c3;
Gyz = c1.*Y.*Z.*c3;



function [cGxy cGyx cGxz cGzx cGyz cGzy] = curlDyadicGreens(k,X,Y,Z)
% Curl of the electromagnetic free space dyadic Green's function
% Cartesian coordinates and vectors

% k:        background wavenumber
% X:        (x-x') argument
% Y:        (y-y') argument
% Z:        (z-z') argument
%           X, Y, and Z are the same size

% [cGxx cGyy cGzz cGxy cGxz cGyz]:
%           6 unique dyadic components of curl G
%           Components are the same size of X, Y, and Z
%           (curl G)_xx = (curl G)_yy = (curl G)_zz = 0

% curl G =  [0        cGxy        cGxz]
%           [cGyx        0        cGyz]
%           [cGzx        cGzy        0]
%
%       =   [0        -d/dz        d/dy]
%           [d/dz        0        -d/dx] * exp(ik r) / (4 pi r)
%           [-d/dy        d/dx        0]

% Straight evaluation, no provisions for r=0

r = sqrt(X.^2 + Y.^2 + Z.^2);
f = (1i*k*r-1).*exp(1i*k*r)./(4*pi*(r.^3));
cGxy = -Z.*f;
cGxz = Y.*f;
cGyz = -X.*f;

% anti-symmetry
cGyx = -cGxy;
cGzx = -cGxz;
cGzy = -cGyz;
```

2.3 Volume Integral Equation

The volume integral equation (VIE) formulates the scattering solution of a heterogeneous distribution of material in terms of the Green's function for homogenous media, [2]. The VIE applies to both scalar and vector problems and is the basis of many numerical solvers, such as the Method of Moments. It serves as the basis for many inverse scattering algorithms.

2.3.1 Scalar VIE

Start with the Helmholtz wave equation where the wavenumber of the medium is a function of position

$$\nabla^2\phi(\mathbf{r}) + k^2(\mathbf{r})\phi(\mathbf{r}) = s(\mathbf{r}) \tag{2.31}$$

This equation is classified as an inhomogeneous, linear, second-order partial differential equation with non-constant coefficients. Assuming that the material object is bounded and sits in a background medium with wavenumber k_b, the next step is to add and subtract k_b^2 from the object as

$$\nabla^2\phi(\mathbf{r}) + (k^2(\mathbf{r}) + k_b^2 - k_b^2)\phi(\mathbf{r}) = s(\mathbf{r}) \tag{2.32}$$

Keeping the factor of $+k_b^2$ on the LHS and moving everything else to the RHS

$$\nabla^2\phi(\mathbf{r}) + k_b^2\phi(\mathbf{r}) = s(\mathbf{r}) - (k^2(\mathbf{r}) - k_b^2)\phi(\mathbf{r}) \tag{2.33}$$

The LHS is the wave equation for the homogenous background which is sourced by everything on the RHS. The field solution is given by integrating the RHS with the free-space Green's function per (2.2)

$$\phi(\mathbf{r}) = -\int g(\mathbf{r},\mathbf{r}')s(\mathbf{r}')dV' + \int g(\mathbf{r},\mathbf{r}')(k^2(\mathbf{r}') - k_b^2)\phi(\mathbf{r}')dV' \tag{2.34}$$

The first term on the RHS is the field due to the source in the absence of the object. This is the incident field, $\phi_{inc}(\mathbf{r})$. The scalar VIE is then written

$$\phi(\mathbf{r}) = \phi_{inc}(\mathbf{r}) + \int g(\mathbf{r},\mathbf{r}')O(\mathbf{r}')\phi(\mathbf{r}')dV' \tag{2.35}$$

where the object function, $O(\mathbf{r}) = k^2(\mathbf{r}) - k_b^2$, is the contrast of the object relative to the background. This is a Fredholm integral equation of the second kind because the total field, $\phi(\mathbf{r})$, appears inside and outside the integral. The integral term is defined as the scattered field. The scattered field modifies the incident field to give the total field. The scattered field exists inside and outside the object, even though the scattered field depends on the total field in the object. We can express this idea simply as

$$\phi(\mathbf{r}) = \phi_{inc}(\mathbf{r}) + \phi_{sca}(\mathbf{r}) \tag{2.36}$$

In experiment, only the incident and total fields can be measured directly using sources and receivers placed away from the object. The incident field is measured in the absence of the object and the total field measured in the presence of the object. The scattered field is obtained by subtracting the measured incident and total fields. For situations in which the object cannot be removed, the incident field has to be predicted using a model of the source and receiver.

2.3.2 Vector VIE

The vector VIE is derived from the vector wave equation in the same way as the scalar VIE is derived from the Helmholtz wave equation. The vector wave equation for the total electric field in an isotropic inhomogeneous non-magnetic dielectric medium is [2]

$$\nabla \times \nabla \times \mathbf{E}(\mathbf{r}) - k^2(\mathbf{r})\mathbf{E}(\mathbf{r}) = i\omega\mu_o\mathbf{J}(\mathbf{r}) \tag{2.37}$$

Adding and subtracting the background wavenumber, k_b^2, to $k^2(\mathbf{r})$, rearranging, and integrating the source terms with the free-space dyadic Green's function, the vector VIE is

$$\mathbf{E}(\mathbf{r}) = \mathbf{E}_{inc}(\mathbf{r}) + \int \overline{\mathbf{G}}(\mathbf{r},\mathbf{r}') \cdot O(\mathbf{r}')\mathbf{E}(\mathbf{r}')dV' \tag{2.38}$$

$\mathbf{E}_{inc}(\mathbf{r})$ is the incident field in the absence of the object and can be computed with (2.9) given a the source distribution $\mathbf{J}(\mathbf{r})$. The object function is

$$\begin{aligned} O(\mathbf{r}) &= k^2(\mathbf{r}) - k_b^2 && (2.39)\\ &= k_o^2\left(\delta\epsilon_r(\mathbf{r}) + i\frac{\delta\sigma(\mathbf{r})}{\epsilon_o\omega}\right) && (2.40)\\ \delta\epsilon_r(\mathbf{r}) &= \epsilon_r(\mathbf{r}) - \epsilon_{rb} && (2.41)\\ \delta\sigma(\mathbf{r}) &= \sigma(\mathbf{r}) - \sigma_b && (2.42) \end{aligned}$$

where $k_b = \omega\sqrt{\mu_o\epsilon_b}$ is the background wavenumber, ϵ_b is the background permittivity, and $k_o^2 = \omega^2\mu_o\epsilon_o$ is the lossless free-space background wavenumber. The complex permittivity is, generally, $\epsilon = \epsilon_o(\epsilon_r + i\sigma/(\omega\epsilon_o))$, where ϵ_r is the real part, σ is the conductivity, and ω the natural frequency, [2]. Using this, one can derive the functions $\delta\epsilon(\mathbf{r})$ and $\delta\sigma(\mathbf{r})$ which are the permittivity and conductivity contrasts relative to the background. Note, the dyadic Green's function is evaluated with the background wavenumber k_b.

Using

$$\mathbf{E}(\mathbf{r}) = \mathbf{E}_{inc}(\mathbf{r}) + \mathbf{E}_{sca}(\mathbf{r}) \tag{2.43}$$

the scattered field is given by the integral term

$$\mathbf{E}_{sca}(\mathbf{r}) = \int \overline{\mathbf{G}}(\mathbf{r},\mathbf{r}') \cdot O(\mathbf{r}')\mathbf{E}(\mathbf{r}')dV' \tag{2.44}$$

2.4 Far-field Born Approximation

We give the classic expression for the vector field volume integral equation under the far-field Born approximation. The Born approximation refers to any instance in which the total field in an object is approximated by the incident field. In the far-field Born approximation, the VIE is simplified with three assumptions:

1. The Born approximation is made for the total field in the object: $\mathbf{E}(\mathbf{r}) = \mathbf{E}_{inc}(\mathbf{r})$,
2. The incident field is a plane wave, $\mathbf{E}_{inc}(\mathbf{r}) = \mathbf{E}_i \exp(i\mathbf{k}_i \cdot \mathbf{r})$,
3. The far-field dyadic Green's function, (2.23), is used in the scattered field direction, $\mathbf{k}_s$:

$$\overline{\mathbf{G}}(\mathbf{r},\mathbf{r}') \approx \left[\overline{\mathbf{I}} - \hat{r}\hat{r}\right]\frac{e^{ikr}}{4\pi r}e^{-i\mathbf{k}_s\cdot\mathbf{r}'}$$

Substituting these into (2.44), the far-field Born approximation for the VIE is

$$\mathbf{E}_{sca}(\mathbf{r}) \approx \frac{e^{ikr}}{4\pi r}\left[\overline{\mathbf{I}} - \hat{r}\hat{r}\right] \cdot \mathbf{E}_i \int O(\mathbf{r}')e^{i(\mathbf{k}_i-\mathbf{k}_s)\cdot\mathbf{r}'}dV' \tag{2.45}$$

The integral is the 3D Fourier transform of the object function, $O(\mathbf{r})$, (2.40), in the wave vector difference domain. There is no multiple scattering or depolarization due to scattering: the incident polarization is simply projected onto the scattered field polarizations based on the geometry of the source and observer. The Born approximation is a weak assumption because it assumes that the total field solution is unaffected by the object. It is only valid when the objects are very small compared to the wavelength or have very low contrast. This expression is the foundation of the k-space mapping between source/receiver direction pairs and the Fourier spectral components of the object, and is the basis of diffraction tomography imaging algorithms, [4, 2].

2.5 Volume-Integrated Free-space Green's Functions

The VIEs need to be discretized for numerical computation, for example, in the Method of Moments or Conjugate Gradient FFT. The discretization can be done with cubic voxels that are small enough so that the total field and object are considered constant in the voxel after which only the Green's function has to be integrated over the voxel. The Green's function, and therefore the integrand, will be singular whenever $\mathbf{r} = \mathbf{r}'$. However, the singularity is integrable in both the scalar and vector cases. When $\mathbf{r} \neq \mathbf{r}'$, the volume-integrated Green's function takes a different but constant value. The solutions to integrating the singularity are analytical if the voxel is spherical, therefore, cubic voxels are replaced with volume-equivalent spheres. The derivations are involved (e.g., principal values, exclusion volumes), but the result is a straightforward replacement of the continuous VIE with a sum over discrete cells and appropriate scale factors.

3D Voxel Integration From [2, 5], the volume-integrated 3D scalar and dyadic Green's functions at the singular points are

$$g(\mathbf{r} = \mathbf{r}') \to \frac{1}{k_b^2}\left(-1 + (1 - ik_b a)e^{ik_b a}\right) \tag{2.46}$$

and

$$\mathbf{G}(\mathbf{r} = \mathbf{r}') \to \frac{1}{k_b^2}\left(-1 + \frac{2}{3}(1 - ik_b a)e^{ik_b a}\right)\bar{\mathbf{I}} \tag{2.47}$$

where a is the radius of the volume-equivalent sphere of the cubic voxels and k_b is the background wavenumber. These values replace the voxel-integrated Green's function at the singularity and no factor of differential volume is needed. The source, object, or total field is assumed constant at the singular point and sampled directly. When substituted into the VIE, the factor of $1/k_b^2$ will cancel the dimensions of the object function, $O(\mathbf{r})$, which has dimensions of k_b^2, leaving only the dimensions of the total field.

For non-singular points, assuming that the observation point is outside of the sphere surrounding the singular point, the Green's function, source, object or total field is sampled directly, but the differential volume is replaced with, [5],

$$\Delta V = \frac{4\pi a}{k_b^2}\left(\frac{\sin(k_b a)}{k_b a} - \cos(k_b a)\right) \tag{2.48}$$

This applies to both scalar and dyadic cases. Note, ΔV has dimensions of length cubed. To gain more physical insight, we can rewrite this as

$$\Delta V = \frac{4}{3}\pi a^3\left(3\frac{\sin(k_b a) - k_b a\cos(k_b a)}{(k_b a)^3}\right) \tag{2.49}$$

The oscillating term in parentheses accounts for the coherence/decoherence of summing the phase term of the Green's function over the sphere. It has a maximum value of 1 when $k_b a \to 0$. This means that as the voxel size decreases, the integration of the Green's function becomes less important, and the volume element can be approximated just as well with the cubic volume. This expression is nearly identical to that derived in Section 6.6 for the volume phase integral over a sphere.

2D Voxel Integration When the volume integrals are 2D, and discretized with squares, the singular point of the 2D scalar Green's function integrates to, [5],

$$g(\boldsymbol{\rho} = \boldsymbol{\rho}') \to -\frac{1}{k_b^2} + \frac{i\pi a}{2k_b}H_1^{(1)}(k_b a) \tag{2.50}$$

This has dimensions of the length squared, which cancels the dimensions of the object function in the VIE. The differential surface element is replaced with

$$\Delta S = \frac{2\pi a}{k_b}J_1(k_b a) \tag{2.51}$$

where a is the radius of the area-equivalent disk, k_b is the background wavenumber and ΔS has dimensions of length squared. Similar to (2.48), (2.51) is equal to the area of the voxel when $k_b a \to 0$.

The routine `volintGreens` returns the volume-integrated Green's function and discrete volume element for the cases above. It takes as input the radius of the circular or spherical voxel, a, the background wavenumber, k_b, and a string switch with three options: `'2D'` for scalar 2D, `'3D'` for scalar 3D, or `'dyadic'` for the 3D dyadic version. Typically, a and k_b are constant in a given VIE discretization, but the routine is vectorized to enable study over a range of values.

```
function [sing delta] = volintGreens(a,kb,str)
% Volume-intergrated scalar and dyadic Green's function
%
% a:            radius of circular or spherical voxel
% kb:           background wavenumber
% str:          '2D'        : 2D scalar VIE
%               '3D'        : 3D scalar VIE
%               'dyadic'    : 3D vector VIE
%
% sing:         Green's function integrated at the singular point
% delta:        Discrete surface or volume element for non-singular points

% singular point
if strcmp(str,'2D') % 2D scalar VIE
    sing = -1./kb.^2 + (1i*pi/2)*(a./kb).*besselh(1,kb.*a);
elseif strcmp(str,'3D') % 3D scalar VIE
    sing = (1./kb.^2).*(-1 + (1 - 1i*kb.*a).*exp(1i*kb.*a));
elseif strcmp(str,'dyadic') % 3D vector VIE
    sing = (1./kb.^2).*(-1 + (2/3)*(1 - 1i*kb.*a).*exp(1i*kb.*a));
else
    error('Bad string')
end

% Discrete surface or volume element for 2D or 3D
if strcmp(str,'2D')
    delta = 2*pi*(a./kb).*besselj(1,kb.*a);
else
    delta = 4*pi*(a./kb.^2).*(sin(kb.*a)./(kb.*a) - cos(kb.*a));
end
```

2.6 Method of Moments

The VIEs provide a framework for solving for the total field solution inside an inhomogeneous distribution of material and then using the total field solution to predict the scattered field at points away from the object. In their continuous forms, the VIEs are nonlinear functions of the total field. The Method of Moments (MoM, or Moment Method) is a procedure to discretize the VIEs in order to cast the scattering problem as a system of linear equations, to which linear algebra algorithms can be applied. In general, for an accurate solution, an object must be discretized better than $\lambda/10$ for the wavelength in the highest dielectric constant in the object.

2.6.1 Pulse Basis Function

The simplest discretization of the VIE are pulse basis functions. The object and field are sampled with non-overlapping cubic voxels, where it is assumed the the object and field are constant in each voxel. The object function is approximated as

$$O(\mathbf{r}) \approx \sum_n O_n \delta_n(\mathbf{r}) \tag{2.52}$$

$$\delta_n(\mathbf{r}) = \begin{cases} 1, & \mathbf{r} \in \text{voxel } n \\ 0, & \text{otherwise} \end{cases} \tag{2.53}$$

where n indexes the set of voxels in 2D or 3D.

2.6.2 Scalar MoM

Using the pulse basis function, two versions of the MoM for the scalar VIE can be derived. The first is written as a solution for the total field. This is useful for inverse scattering problems when the total field is needed separate from the object. The second version is written in terms of the induced source, or contrast source, [6]. The induced source is advantageous if a) the total field is not needed, b) there are many objects to simulate, because the object only appears the diagonal of the MoM matrix, or c) only the scattered field away from the object needs to be computed.

Total Field Substituting the pulse basis function discretization of the object, (2.53), into the VIE, (2.35)

$$\phi(\mathbf{r}) = \phi_{inc}(\mathbf{r}) + \sum_n \int g(\mathbf{r}, \mathbf{r}') O_n \delta_n(\mathbf{r}') \phi(\mathbf{r}') dV' \tag{2.54}$$

Next, the incident and total fields are assumed constant in each voxel and the fields outside of the integral are sampled as the same points as those in the integrand, but indexed separately

$$\phi(\mathbf{r}_m) = \phi_{inc}(\mathbf{r}_m) + \sum_n \int g(\mathbf{r}_m, \mathbf{r}') O_n \delta_n(\mathbf{r}') \phi(\mathbf{r}_n) dV' \tag{2.55}$$

The last step is to integrate the Green's function over the voxels. Using the results of Section (2.5), this becomes

$$\phi_m = \phi_{inc,m} + \sum_n g_{mn} O_n \phi_n \tag{2.56}$$

$$g_{mn} = \begin{cases} g(\mathbf{r}_m, \mathbf{r}_n) \Delta V, & m \neq n \\ g(\mathbf{r}_m = \mathbf{r}_n), & m = n \end{cases} \tag{2.57}$$

where ΔV is given by (2.48), and the integration over the singular point is given by (2.46), where the cubic voxels have been replaced by volume-equivalent spheres. The same procedure applies in 2D.

Writing (2.56) in matrix notation

$$(\mathbf{I} - \mathbf{GO})\, \boldsymbol{\phi} = \boldsymbol{\phi}_{inc} \tag{2.58}$$

where $\mathbf{I}$ is the identity matrix, $\boldsymbol{\phi}$ is the vector of unknown total field values at each voxel, $\boldsymbol{\phi}_{inc}$ is the known incident field at the same voxels, $\mathbf{G}$ is a full, symmetric matrix containing the Green's function values, and $\mathbf{O}$ is a diagonal matrix containing the values of the object function at each voxel. The right multiplication of $\mathbf{O}$ to $\mathbf{G}$ makes the final matrix asymmetric for inhomogeneous objects. A zero-contrast voxel has the effect of zeroing-out a column of the Green's function matrix, but the identity matrix makes the overall MoM matrix safe to invert. This last feature is one reason why the total field formulation is useful for iterative inverse scattering problems where the object contrast is not known ahead of time. Put another way, when the object contrast is zero, the total field solution is not necessarily zero.

Induced Source The induced source, or contrast source, is defined as the product of the object contrast and the total field

$$w(\mathbf{r}) = O(\mathbf{r})\phi(\mathbf{r}) \tag{2.59}$$

Multiplying (2.35) by $O(\mathbf{r})$, the VIE can be written

$$w(\mathbf{r}) = w_{inc}(\mathbf{r}) + O(\mathbf{r}) \int g(\mathbf{r}, \mathbf{r}')w(\mathbf{r}')dV' \tag{2.60}$$

Applying the MoM, we get the linear system

$$(\mathbf{I} - \mathbf{OG})\,\boldsymbol{w} = \boldsymbol{w}_{inc} \tag{2.61}$$

This is similar to (2.58) where the LHS matrix is asymmetric. Alternatively, taking (2.60) or (2.61) and multiplying through by $1/O(\mathbf{r})$ we can write

$$\left(\mathbf{O}^{-1} - \mathbf{G}\right)\boldsymbol{w} = \boldsymbol{\phi}_{inc} \tag{2.62}$$

In this form the object only appears along the diagonal which makes the entire LHS matrix symmetric. This is similar to the more standard representation of the MoM, [7], when it is given in terms of the impedance matrix, the induced current, and the incident field. In (2.62), if the object contrast of a voxel is zero, then the diagonal matrix element is infinite. The matrix either has to be reformed to exclude the rows and columns of the zero contrast, or a large numerical value needs to be substituted for the inverse contrast. In general, (2.62) is useful for simulating sparse dielectric objects, such as snow aggregates or vegetation, because the MoM matrix only needs to contain the interactions between voxels that have non-zero contrast.

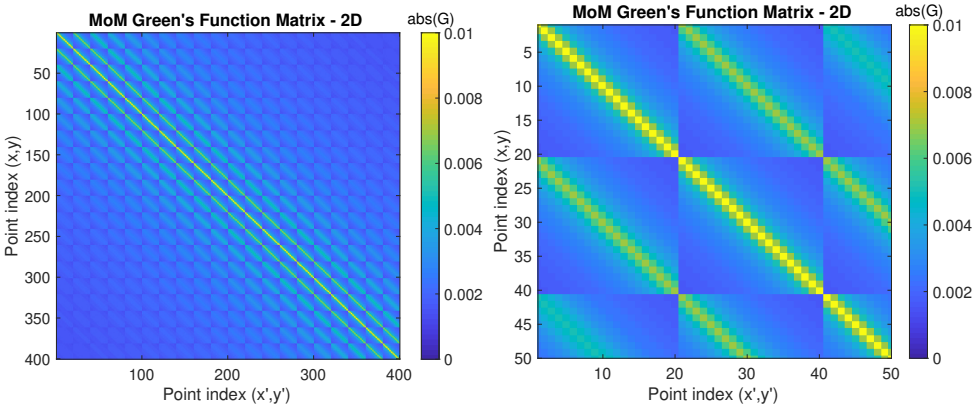


Figure 2.1: 2D scalar Green's function matrix, $\mathbf{G}$. Left: absolute value of the elements for the full matrix for a 20×20 grid of points sampled at $\lambda/10$ (total size $2\lambda \times 2\lambda$). Right: zoom in. The blocks are due to vectorizing the 2D grid of points and then taking all possible pairs of interactions. For example, the upper left 20×20 block contains the 20^2 interactions between the 20 points along one edge of the 2D grid. The diagonal of the matrix contains the self terms.

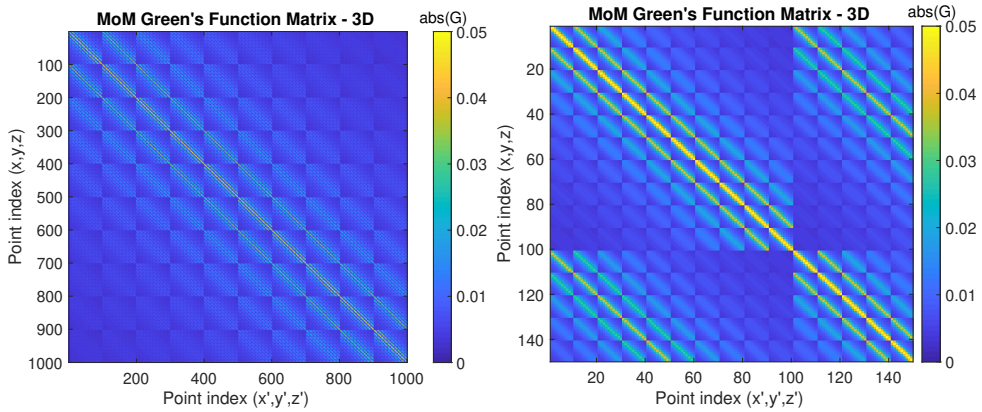


Figure 2.2: 3D scalar Green's function matrix, $\mathbf{G}$. Left: full matrix for a $10 \times 10 \times 10$ grid of points sampled at $\lambda/5$ (total size of $2\lambda \times 2\lambda \times 2\lambda$). Note, this discretization step is for illustration purposes and is insufficient for an accurate solution. Right: zoom in. The appearance of blocks and subblocks is due to first vectorizing the 3D grid of points, and then taking all possible pairs of interactions which are arranged across rows and columns. For example, the first upper left subblock (sized 10×10), contains the 100 pairs of interactions between the 10 points along one edge of the 3D grid. The diagonal of the matrix are the self terms.

Routines The routine `momGmatrix2D` returns the 2D scalar Green's function matrix for the MoM solution. It takes as input the side length of the square voxel, Δx, which is assumed constant for all voxels, the background wavenumber k_b, and the (x, y) and (x', y') coordinate pairs at which the Green's function will be evaluated. The voxel size length is used to evaluate the volume-integrated singular and non-singular scalar factors which are automatically included in the matrix elements. The arrays of unprimed and primed coordinate pairs can be different sizes: the output matrix is sized $M \times N$ where M is the total number of unprimed points down rows, and N is the total number of primed points across columns. The inputs of primed and unprimed coordinates are separated to facilitate the construction of $\mathbf{G}$ in subblocks when the number of points is large. If the complete matrix is desired, just input the same arrays for unprimed and primed coordinates, and the routine will return the full square matrix. The points are vectorized column-wise, so the corresponding object function or incident field need to be vectorized column-wise to be consistent.

```
function G = momGmatrix2D(dx,kb,X,Y,Xp,Yp)
% Create the 2D scalar MoM Green's function matrix
%
% dx:       side length of the square voxel (assumed constant)
% kb:       background free-space wavenumber
% X,Y:      (x,y) coordinates, same size
% Xp,Yp:    (x',y') coordinates, same size
%
% G:        2D scalar MoM Green's function matrix
%           size: MxN, M = length(X(:)), N = length(Xp(:))
%
% Dependencies: volintGreens

% radius of area-equivalent circle of square voxel
a = dx/sqrt(pi);

% get the volume integrated voxels
[sing delta] = volintGreens(a,kb,'2D');

% make primed and unprimed coordinate pairs
[XX XXP] = ndgrid(X(:),Xp(:));
[YY YYP] = ndgrid(Y(:),Yp(:));
Rho = sqrt((XX - XXP).^2 + (YY - YYP).^2);

% evaluate 2D scalar Green's function
G = (delta*1i/4)*besselh(0,kb*Rho);

% replace any self terms with the integrated singularity
G(Rho == 0) = sing;
```

The routine `momGmatrix3D` returns the 3D scalar Green's function matrix for the MoM solution. It takes as input the side length of the cubic voxel, Δx, which is assumed constant for all voxels, the background wavenumber k_b, and the (x, y, z) and (x', y', z') coordinate pairs at which to evaluate the matrix. It otherwise works the same as `momGmatrix2D`. Because the MoM matrix scales as $O(N^2)$ with the number of 3D voxels, N, the total number of elements scales as $O(L^6)$, where L is the side length of a cubic simulation region.

```
function G = momGmatrix3D(dx,kb,X,Y,Z,Xp,Yp,Zp)
% Create the 3D scalar MoM Green's function matrix
%
% dx:       side length of the cubic voxel (assumed constant)
% kb:       background free-space wavenumber
% X,Y,Z:    (x,y,z) coordinates, same size
% Xp,Yp,Zp: (x',y',z') coordinates, same size
%
% G:        3D scalar MoM Green's function matrix
%           size: MxN, M = length(X(:)), N = length(Xp(:))
%
% Dependencies: volintGreens

% radius of volume-equivalent sphere of cubic voxel
a = dx/(3/(4*pi))^(1/3);

% get the volume integrated voxels
[sing delta] = volintGreens(a,kb,'3D');

% make primed and unprimed coordinate pairs
[XX XXP] = ndgrid(X(:),Xp(:));
[YY YYP] = ndgrid(Y(:),Yp(:));
[ZZ ZZP] = ndgrid(Z(:),Zp(:));
R = sqrt((XX - XXP).^2 + (YY - YYP).^2 + (ZZ - ZZP).^2);

% evaluate 3D scalar Green's function
G = delta*exp(1i*kb*R)./(4*pi*R);

% replace any self terms with the integrated singularity
G(R == 0) = sing;
```

2.6.3 Vector MoM

The vector MoM is derived the same way as the scalar MoM. The major difference is in bookkeeping the vector components and the size of the final matrix. The full vector formulation is needed for any 3D vector scattering problem.

Total Field Starting with the vector VIE, (2.38), and substituting the pulse basis function

$$\mathbf{E}(\mathbf{r}) = \mathbf{E}_{inc}(\mathbf{r}) + \sum_n \int \overline{\mathbf{G}}(\mathbf{r}, \mathbf{r}') \cdot O_n \delta_n(\mathbf{r}') \mathbf{E}(\mathbf{r}') dV' \tag{2.63}$$

As with the scalar case, the field is assumed constant in each voxel and indexed as

$$\mathbf{E}(\mathbf{r}_m) = \mathbf{E}_{inc}(\mathbf{r}_m) + \sum_n \int \overline{\mathbf{G}}(\mathbf{r}_m, \mathbf{r}') \cdot O_n \delta_n(\mathbf{r}') \mathbf{E}(\mathbf{r}_n) dV' \tag{2.64}$$

Assuming Cartesian vector components, $u = (x, y, z)$ and $v = (x, y, z)$, and integrating the dyadic Green's function over the volume-equivalent sphere of the cubic voxel, this is written as a sum for each total field component as

$$E_{u,m} = E_{inc,u,m} + \sum_v \sum_n G_{uv,mn} O_n E_{v,n} \tag{2.65}$$

$$G_{uv,mn} = \begin{cases} G_{uv}(\mathbf{r}_m, \mathbf{r}_n) \Delta V, & m \neq n \\ G_{uv}(\mathbf{r}_m = \mathbf{r}_n), & m = n, u = v \quad (0, u \neq v) \end{cases} \tag{2.66}$$

where ΔV is given by (2.48). The value of the singular integration at the self term, (2.47), is equal to a constant times the identity dyad. The identity dyad means that the self-term only applies to the diagonal elements of the dyad (i.e., G_{xx}, G_{yy}, G_{zz}) and is zero for the off-diagonal elements of the dyad. Writing this in matrix notation for the three unknown total field components

$$\left(\mathbf{I} - \begin{bmatrix} \mathbf{G}_{xx} & \mathbf{G}_{xy} & \mathbf{G}_{xz} \\ \mathbf{G}_{yx} & \mathbf{G}_{yy} & \mathbf{G}_{yz} \\ \mathbf{G}_{zx} & \mathbf{G}_{zy} & \mathbf{G}_{zz} \end{bmatrix} \begin{bmatrix} \mathbf{O} & & \\ & \mathbf{O} & \\ & & \mathbf{O} \end{bmatrix} \right) \begin{bmatrix} \mathbf{E}_x \\ \mathbf{E}_y \\ \mathbf{E}_z \end{bmatrix} = \begin{bmatrix} \mathbf{E}_{inc,x} \\ \mathbf{E}_{inc,y} \\ \mathbf{E}_{inc,z} \end{bmatrix} \tag{2.67}$$

where $\mathbf{I}$ is the identity matrix, $\mathbf{G}_{uv}$ is the Green's function matrix block for a given polarization pair, $\mathbf{E}_{inc,v}$ and $\mathbf{E}_v$ are vectors of the incident field and total field solution, respectively, $\mathbf{O}$ is the diagonal matrix of the object function which is applied to each component. The self term (diagonals) of $\mathbf{G}_{xx}, \mathbf{G}_{yy}, \mathbf{G}_{zz}$, take the value, (2.47), while the self terms of the other blocks are zero. Recall, $\mathbf{G}_{uv} = \mathbf{G}_{vu}$, and each block matrix is square symmetric. The size of each block matrix and field vector is the same size as the scalar case. The fields components lead to 3 times as many unknowns as compared to the scalar cases, and polarization mixing makes the MoM matrix 9 times larger than the 3D scalar case.

Induced Current Define the induced current as the product of the contrast and the total field at a given point

$$\mathbf{J}(\mathbf{r}) = O(\mathbf{r})\mathbf{E}(\mathbf{r}) \tag{2.68}$$

Then (2.67) can be written

$$\left(\begin{bmatrix} \mathbf{O}^{-1} & & \\ & \mathbf{O}^{-1} & \\ & & \mathbf{O}^{-1} \end{bmatrix} - \begin{bmatrix} \mathbf{G}_{xx} & \mathbf{G}_{xy} & \mathbf{G}_{xz} \\ \mathbf{G}_{yx} & \mathbf{G}_{yy} & \mathbf{G}_{yz} \\ \mathbf{G}_{zx} & \mathbf{G}_{zy} & \mathbf{G}_{zz} \end{bmatrix} \right) \begin{bmatrix} \mathbf{J}_x \\ \mathbf{J}_y \\ \mathbf{J}_z \end{bmatrix} = \begin{bmatrix} \mathbf{E}_{inc,x} \\ \mathbf{E}_{inc,y} \\ \mathbf{E}_{inc,z} \end{bmatrix} \tag{2.69}$$

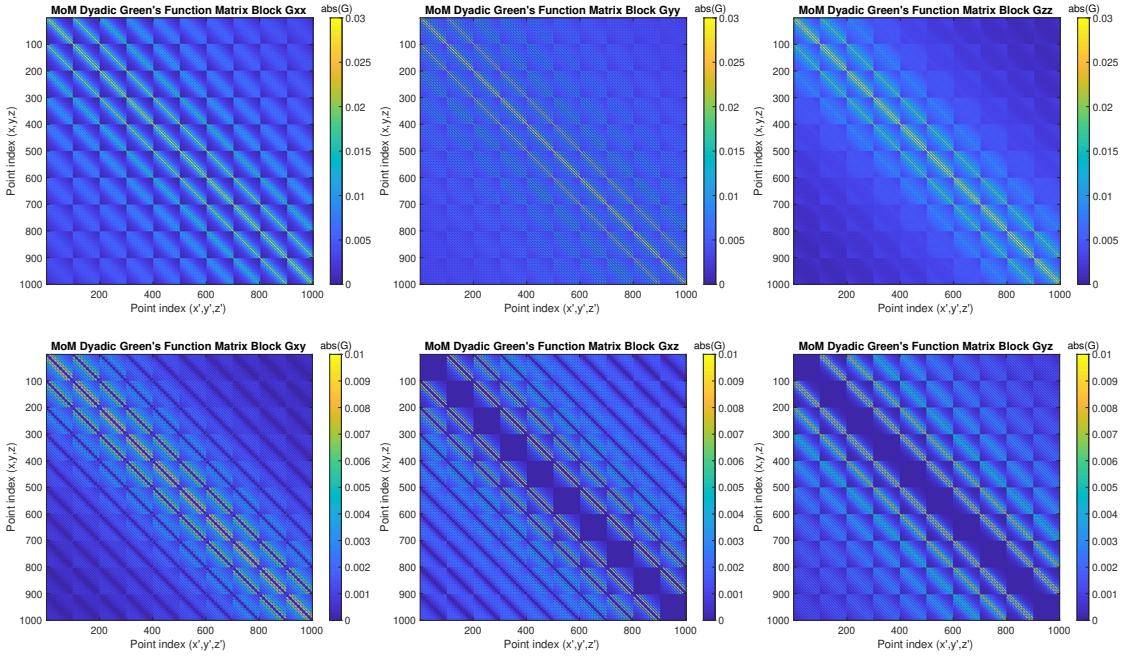


Figure 2.3: 3D dyadic Green's function block matrices, $\mathbf{G}_{uv}$. Full block matrices for a $10 \times 10 \times 10$ grid of points sampled at $\lambda/5$ (total size of $2\lambda \times 2\lambda \times 2\lambda$). The zero elements in $\mathbf{G}_{u \neq v}$ happen when voxels have common coordinates in either u or v. This is a result of $\mathbf{G}_{u \neq v}$ being proportional to the difference of primed and unprimed coordinates in u or v.

Routine The routine `momGmatrixDyadic` returns the 6 unique block matrices of the 3D dyadic MoM matrix. It takes as input the side length of the cubic voxel, Δx, which is assumed constant for all voxels, the background wavenumber k_b, and the (x, y, z) and (x', y', z') coordinate pairs at which to evaluate the matrix. It works similarly to `momGmatrix2D` and `momGmatrix3D`, but calls `dyadicGreens` which takes the relative positions as input. The arrays of unprimed and primed coordinate pairs can be different sizes. The output matrix is sized $M \times N$ where M is the total number of unprimed points down rows, and N is the total number of primed points across columns. In the routine, the volume integrated constants are applied to all entries. The singularity constant is applied to the self terms of the blocks $\mathbf{G}_{u=v}$, while the self terms of the blocks $\mathbf{G}_{u\neq v}$ are set to zero.

```
function [Gxx Gyy Gzz Gxy Gxz Gyz] = momGmatrixDyadic(dx,kb,X,Y,Z,Xp,Yp,Zp)
% Create the dyadic MoM Green's function matrix
%
% dx:       side length of the cubic voxel (assumed constant)
% kb:       background free-space wavenumber
% X,Y,Z:    (x,y,z) coordinates, same size
% Xp,Yp,Zp: (x',y',z') coordinates, same size
%
% [Gxx Gyy Gzz Gxy Gxz Gyz]:    6 unique matrix blocks of the 3D dydaic MoM Green's
%                               Block size: MxN, M = length(X(:)), N = length(Xp(:))
%
% Dependencies: volintGreens, dyadicGreens

% radius of volume-equivalent sphere of cubic voxel
a = dx/(3/(4*pi))^(1/3);

% get the volume integrated voxels
[sing delta] = volintGreens(a,kb,'dyadic');

% make primed and unprimed coordinate pairs
[XX XXP] = ndgrid(X(:),Xp(:));
[YY YYP] = ndgrid(Y(:),Yp(:));
[ZZ ZZP] = ndgrid(Z(:),Zp(:));
dX = (XX - XXP);
dY = (YY - YYP);
dZ = (ZZ - ZZP);
clear XX XXP YY YYP ZZ ZZP   % ...to save some space

% evaluate 3D dyadic Green's function
[Gxx Gyy Gzz Gxy Gxz Gyz] = dyadicGreens(kb,dX,dY,dZ);

% multiply by the volume factor
Gxx = delta*Gxx;
Gyy = delta*Gyy;
Gzz = delta*Gzz;
Gxy = delta*Gxy;
Gxz = delta*Gxz;
Gyz = delta*Gyz;

% replace any self terms in Gxx, Gyy, Gzz with the integrated singularity
R = sqrt(dX.^2 + dY.^2 + dZ.^2);
ind = find(R==0);
Gxx(ind) = sing;
Gyy(ind) = sing;
Gzz(ind) = sing;

% replace any self terms in Gxy, Gxz, Gyz with zero
Gxy(ind) = 0;
Gxz(ind) = 0;
Gyz(ind) = 0;
```

2.7 Scattered Field VIE

Once the total field solution is found, the scattered field away from the object is computed again with the VIE. Here, the unprimed position vector of the Green's function is evaluated at an observation point outside of the object domain. In this case, the Green's function is often referred to as the receiver Green's function. This can be a point of confusion when comparing it to the Green's function as used in the MoM solution, where the unprimed position vector is evaluated throughout the object domain, including at singular points. In the end, the receiver Green's function is the same as the one used in the MoM solution, except that the unprimed position vector is simply evaluated outside of the object at a point that is also usually associated with a sensor.

From reciprocity, it turns out that the receiver Green's function is equivalent to the incident field generated by a point source at the observation location. This is practically and conceptually useful. It links the incident field needed when solving for the total field solution to the scattered field VIE needed to compute the measurements from sources or receivers at the same location. This idea is generalized to the case of real antennas fed by waveguides using the waveport vector Green's function described in [8]. The overarching idea here is that only the incident field of a source or an antenna needs to be known in order to compute both the total field solution and the received voltage measurements in a consistent way.

2.7.1 Receiver Green's Function

Scalar case Let a source be at location $\mathbf{r}_i$. This generates an incident field, $\phi_{inc,i}(\mathbf{r})$, for which the total field solution in the object is $\phi_i(\mathbf{r})$. The scattered field observed at a point, $\mathbf{r}_j$, outside the object is

$$\phi_{sca,ji}(\mathbf{r}_j) = \int g(\mathbf{r}_j, \mathbf{r}')O(\mathbf{r}')\phi_i(\mathbf{r}')dV', \qquad \mathbf{r}_j \notin V \tag{2.70}$$

where, $g(\mathbf{r}_j, \mathbf{r}')$ is the receiver Green's function. The receiver Green's function is named this way because the unprime variable is evaluated at the receiver, or observation, point. This VIE is discretized the same way as the MoM: the Green's function is integrated over volume-equivalent voxels so that

$$\phi_{sca,ji}(\mathbf{r}_j) \approx \Delta V \sum_n g(\mathbf{r}_j, \mathbf{r}_n)O(\mathbf{r}_n)\phi_i(\mathbf{r}_n) \tag{2.71}$$

where ΔV is given by (2.48) in the 3D case. Because $\mathbf{r}_j \notin V$, there is no singular point to deal with.

Vector case Let a vector source, like an antenna, be located at $\mathbf{r}_i$. This generates an incident field, $\mathbf{E}_{inc,i}(\mathbf{r})$, for which the total field solution in the object is $\mathbf{E}_i(\mathbf{r})$. The scattered field observed at a point, $\mathbf{r}_j$, outside the object is

$$\mathbf{E}_{sca,ji}(\mathbf{r}_j) = \int \overline{\mathbf{G}}(\mathbf{r}_j, \mathbf{r}') \cdot O(\mathbf{r}')\mathbf{E}_i(\mathbf{r}')dV', \qquad \mathbf{r}_j \notin V \tag{2.72}$$

Here, $\overline{\mathbf{G}}(\mathbf{r}_j, \mathbf{r}')$ is the dyadic receiver Green's function. Discretized like the MoM, this is

$$\mathbf{E}_{sca,ji}(\mathbf{r}_j) \approx \Delta V \sum_n \overline{\mathbf{G}}(\mathbf{r}_j, \mathbf{r}_n) \cdot O(\mathbf{r}_n)\mathbf{E}_i(\mathbf{r}_n) \tag{2.73}$$

where ΔV is given by (2.48), there is no singular point, and the dyadic Green's function is evaluated normally.

2.7.2 Incident Field Reciprocity

The receiver Green's function is equivalent to the incident field generated by a point source at the receiver location when used in transmit mode. This is a consequence of the definition of the Green's function and reciprocity. The derivation is simple, but the idea has important implications for linking the VIE to measurements in experimental setups. In short, the incident field alone fully characterizes the transmit and receive characteristics of a reciprocal sensor. We derive the results for scalar and vector point sources, but the results hold for arbitrary source distributions.

Scalar case Define a scalar point source at location $\mathbf{r}_j$ as $s(\mathbf{r}) = -\delta(\mathbf{r}-\mathbf{r}_j)$. Using (2.2), the incident field from this source is

$$\begin{aligned}\phi_{inc,j}(\mathbf{r}) &= -\int g(\mathbf{r},\mathbf{r}')s(\mathbf{r}')dV' &(2.74)\\ &= \int g(\mathbf{r},\mathbf{r}')\delta(\mathbf{r}'-\mathbf{r}_j)dV' &(2.75)\\ &= g(\mathbf{r},\mathbf{r}_j) &(2.76)\end{aligned}$$

This is another way of stating the definition of the Green's function. Using the fact that $g(\mathbf{r},\mathbf{r}_j) = g(\mathbf{r}_j,\mathbf{r})$, and substituting this into (2.70), the scattered field VIE is

$$\phi_{sca,ji}(\mathbf{r}_j) = \int \phi_{inc,j}(\mathbf{r}')O(\mathbf{r}')\phi_i(\mathbf{r}')dV' \tag{2.77}$$

The incident field from the point source takes the place of the receiver Green's function. From linearity, this holds for an incident field from an arbitrary source distribution so long as the same device is used as the receiver. Finally, (2.77) is discretized the same as (2.71).

Vector case Let a current source be an infinitesimal dipole at location $\mathbf{r}_j$ with strength I and polarization $\hat{\mathbf{p}}$

$$\mathbf{J}(\mathbf{r}) = I\hat{\mathbf{p}}\delta(\mathbf{r}-\mathbf{r}_j) \tag{2.78}$$

Substituting this into (2.9)

$$\begin{aligned}\mathbf{E}_{inc,j}(\mathbf{r}) &= i\omega\mu\int \overline{\mathbf{G}}(\mathbf{r},\mathbf{r}')\cdot\mathbf{J}(\mathbf{r}')dV' &(2.79)\\ &= i\omega\mu I\int \overline{\mathbf{G}}(\mathbf{r},\mathbf{r}')\cdot\hat{\mathbf{p}}\delta(\mathbf{r}'-\mathbf{r}_j)dV' &(2.80)\\ &= i\omega\mu I\overline{\mathbf{G}}(\mathbf{r},\mathbf{r}_j)\cdot\hat{\mathbf{p}} &(2.81)\end{aligned}$$

or

$$\overline{\mathbf{G}}(\mathbf{r},\mathbf{r}_j)\cdot\hat{\mathbf{p}} = \frac{1}{i\omega\mu I}\mathbf{E}_{inc,j}(\mathbf{r}) \tag{2.82}$$

A similar expression can be found in [9] for 2D problems. The columns of the dyadic receiver Green's function are found by computing the incident field due to three orthogonal dipoles in turn. Denote these incident fields $\mathbf{E}_{inc,j,p}$ for dipole polarizations $p = [x,y,z]$ located at the observation point. Then using the fact that $\overline{\mathbf{G}}(\mathbf{r},\mathbf{r}_j) = \left[\overline{\mathbf{G}}(\mathbf{r}_j,\mathbf{r})\right]^t$, we have

$$\overline{\mathbf{G}}(\mathbf{r}_j,\mathbf{r}) = \frac{1}{i\omega\mu I}\begin{bmatrix}\mathbf{E}^t_{inc,j,x}(\mathbf{r})\\ \mathbf{E}^t_{inc,j,y}(\mathbf{r})\\ \mathbf{E}^t_{inc,j,z}(\mathbf{r})\end{bmatrix} \tag{2.83}$$

The matrix on the right hand side is denoted the incident field dyad. The transposes place the $[x,y,z]$ vector components of each dipole incident field along the rows of the dyad, which maps the three-vector total field (or induced source) in the domain to the polarizations of the source dipole at the receiver location. Substituting this into (2.72), the scattered field VIE for the electric field is

$$\mathbf{E}_{sca,ji}(\mathbf{r}_j) = \frac{1}{i\omega\mu I}\int \begin{bmatrix}\mathbf{E}^t_{inc,j,x}(\mathbf{r}')\\ \mathbf{E}^t_{inc,j,y}(\mathbf{r}')\\ \mathbf{E}^t_{inc,j,z}(\mathbf{r}')\end{bmatrix}\cdot O(\mathbf{r}')\mathbf{E}_i(\mathbf{r}')dV' \tag{2.84}$$

In simulation, one can set $I = 1/i\omega\mu$ to cancel the scale factor and (2.84) can be discretized like (2.73). The physical interpretation of (2.84) is the following. The three source dipoles at the observation location independently map to a three-vector incident field in the object domain. Concurrently, each vector component of the induced current in the domain radiates all three vector components to the observation point. The columns of the incident field dyad, (2.83), map any single component of the induced current to the three components at the observation location in proportion to the strength of the incident fields created by the corresponding source dipoles. This is another way of stating reciprocity.

2.7.3 Waveport Vector Green's Function

When measuring fields with real antennas, we never measure the three electric field components directly. We measure a voltage on a feeding waveguide or transmission line. The three vector components of the scattered field are effectively integrated over the surface of the antenna and produce the voltage on the waveguide. The incident field dyad above collapses to three-vector incident field with certain scale factors. This was formalized in [8] in the form a waveport vector Green's function. The idea is fundamentally the same as the receiver Green's function and incident field dyad, except that it is specialized for S-parameter measurements between two antennas that are made using a vector network analyzer (VNA) with calibrated reference planes on the feeding transmission lines.

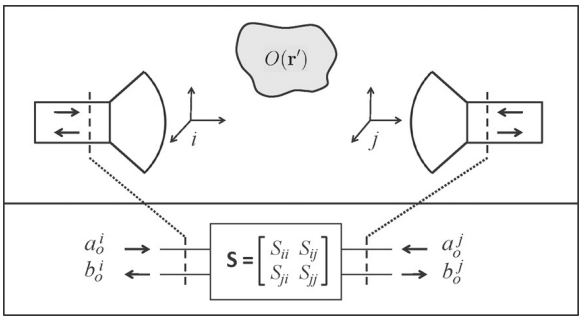


Figure 2.4: Network model of two antennas and a scattering object. S-parameters are measured between the reference planes on antenna transmission lines, [8].

Let two antennas in frames i and j be used to probe an object with a VNA that measures the entire system as a 2-port device, as shown in Figure 2.4. The complex excitation amplitudes on each transmission line are a_o^i and a_o^j, while the received amplitudes are b_o^i and b_o^j. The VNA is calibrated to the reference planes on the transmission lines. Assume that the spatial distribution of the antenna incident field is referenced to a fixed coordinate origin of the antenna, and the excitation (amplitude and phase) is referenced to the calibration planes on the transmission line. The waveport vector Green's function, which replaces the dyadic receiver Green's function, and effectively collapses the incident field dyad in (2.83), is

$$\mathbf{g}(\mathbf{r}) = -\frac{Z_o}{2a_o}\frac{1}{i\omega\mu}\mathbf{E}_{inc}(\mathbf{r}) \tag{2.85}$$

where Z_o is the characteristic impedance of the receiver transmission line and a_o is the excitation used to create $\mathbf{E}_{inc}(\mathbf{r})$. When used in the VIE, the 2-port scattered field S-parameter measurement, S_{ji}, is given by

$$S_{ji} = -\frac{1}{2i\omega\mu}\frac{Z_o^j}{a_o^j a_o^i}\int \mathbf{E}_{inc,j}(\mathbf{r}')\cdot O(\mathbf{r}')\mathbf{E}_i(\mathbf{r}')dV' \tag{2.86}$$

where a_o^j is the excitation used to create the incident field of the receiver, $\mathbf{E}_{inc,j}(\mathbf{r}')$, and a_o^i is the excitation used to create the incident field of the transmitter $\mathbf{E}_{inc,i}(\mathbf{r}')$ which in turn is used to compute the solution of the total field $\mathbf{E}_i(\mathbf{r}')$. In practice, if we know the average transmit power on the transmission line, P_{ave}, then from transmission line analysis the magnitude of a_o is given by

$$|a_o| = \sqrt{2Z_o P_{ave}} \tag{2.87}$$

The phase of a_o can be found by comparing the transmission line reference plane used to measure or simulate the incident fields to the reference planes used in measurement.

Chapter 3

Spherical Wave Functions

This chapter gives routines and tools for computing and using scalar and vector wave functions in free-space spherical coordinates. The wave functions represent the frequency-domain multipoles of the field, which together give the multipole expansion of a field. While the wave functions themselves are not always used, the coefficients of the expansions are quite useful. The expansion coefficients can be quickly manipulated by rotation and translation addition theorems to transform the fields in different reference frames (see Chapters 4 and 5). This chapter provides codes for spherical wave functions, and serves as a quick reference for certain properties and derivations.

3.1 Indexing

Spherical harmonics and spherical wave functions are defined by the degree and order of the underlying associated Legendre polynomials, (l, m). It is convenient to store and access harmonics using a linear index. For vector wave functions, where the harmonics range from $l = 1, ..., L$ and $m = \pm l$, the linear index for harmonic (l, m) is given by $n = l^2 + l + m$. When the set of harmonics includes all m at each l, the total number of harmonics is $N = L^2 + 2L$. The following sequence converts a linear index n to (l, m) for vector wave functions:

$$l = \lfloor \sqrt{n} \rfloor \tag{3.1}$$
$$m = n - l^2 - l \tag{3.2}$$

Table 3.1 illustrates the mapping between linear index and harmonic for vector waves. A similar table appears in [10]. Scalar wave functions include degree $l = 0$, which is the monopole. The linear index is then given by $n = l^2 + l + m + 1$. When the set of harmonics includes all m at each l, the total number of harmonics is $N = L^2 + 2L + 1$. The following sequence converts a linear index n to (l, m) for scalar waves:

$$l = \lfloor \sqrt{n-1} \rfloor \tag{3.3}$$
$$m = n - l^2 - l - 1 \tag{3.4}$$

Table 3.2 illustrates the mapping for scalar wave functions including the monopole.

The function `lm2ind` returns the linear index for pair (l, m), while `ind2lm` outputs the (l, m) pair given the index. Use the string switch `'mono'` to include the monopole for scalar waves. `lmtable` produces a table of (l, m) pairs. These routines are useful for general purpose manipulation of spherical harmonic indexing or when developing routines. However, the error checking slows them down and should be replaced by inline expressions once routines are debugged.

Table 3.1: Vector Wave Function Harmonic Indexing

Linear index	l	m
1	1	-1
2	1	0
3	1	1
4	2	-2
5	2	-1
6	2	0
7	2	1
8	2	2
9	3	-3
10	3	-2
11	3	-1
12	3	0
13	3	1
14	3	2
15	3	3

Table 3.2: Scalar Wave Function Harmonic Indexing

Linear index	l	m
1	0	0
2	1	-1
3	1	0
4	1	1
5	2	-2
6	2	-1
7	2	0
8	2	1
9	2	2
10	3	-3
11	3	-2
12	3	-1
13	3	0
14	3	1
15	3	2
16	3	3

```
function [ind] = lm2ind(l,m,str)
% Returns linear index for anglular harmonic number (l,m)
%
% l,m:  angular harmonic number
% str:  [optional] 'mono' for scalar wave function indexing
%
% ind:  linear index

if nargin == 3
    if ~strcmp(str,'mono')
        error('bad string')
    end
    if sum(or(l<0,abs(m)>l))>0
        error('bad index')
    end
    ind = l.^2 + l + m + 1;
else
    if sum(or(l<1,abs(m)>l))>0
        error('bad index')
    end
    ind = l.^2 + l + m;
end


function [l, m] = ind2lm(ind,str)
% Returns angular harmonic number (l,m) from linear index
%
% ind:  linear index
% str:  [optional] 'mono' for scalar wave function indexing
%
% l,m:  angular harmonic number

if sum(ind<1)>0
    error('bad index')
end
if nargin == 2
    if ~strcmp(str,'mono')
        error('bad string')
    end
    ind = ind - 1;
end
l = floor(sqrt(ind));
```

```
m = ind - l.^2 - l;


function [tab] = lmtable(L,str)
% Table of indices for vector spherical harmonics
%
% L:    Maximum harmonic l
% str:  [optional] 'mono' for scalar wave function indexing
%
% tab:  2 column array with (l,m)
%       (l,m) = (column 1, column 2)

if L < 0
    error('bad index')
end
tot = L^2 + 2*L;
startl = 1;
if nargin == 2
    if ~strcmp(str,'mono')
        error('bad string')
    end
    tot = tot + 1;
    startl = 0;
end
tab = zeros(tot,2);
cnt = 1;
for l=startl:L,
for m=(-l):l,
    tab(cnt,1) = l;
    tab(cnt,2) = m;
    cnt = cnt+1;
end
end
```

3.2 Spherical Harmonics

3.2.1 $Y_{lm}(\theta,\phi)$

The fully normalized spherical harmonics are given by

$$Y_{lm}(\theta,\phi)=\sqrt{\frac{2l+1}{4\pi}\frac{(l-m)!}{(l+m)!}}P_l^m(\cos\theta)e^{im\phi} \tag{3.5}$$

where $P_l^m(\cos\theta)$ are the associated Legendre polynomials. The Condon-Shortley phase, $(-1)^m$, is included in the definition of $P_l^m(\cos\theta)$. The spherical harmonics can be written in terms of the fully normalized associated Legendre polynomials, $\widetilde{P}_l^m(\cos\theta)$, as

$$Y_{lm}(\theta,\phi)=\frac{1}{\sqrt{2\pi}}\widetilde{P}_l^m(\cos\theta)e^{im\phi} \tag{3.6}$$

Any scalar spherical function can be expanded as a sum of spherical harmonics

$$f(\theta,\phi)=\sum_{l=0}^{\infty}\sum_{m=-l}^{l}a_{lm}Y_{lm}(\theta,\phi) \tag{3.7}$$

Being fully normalized, the orthogonality relation for the spherical harmonics is simply

$$\int_0^{2\pi}\int_0^{\pi}Y_{lm}(\theta,\phi)Y^*_{l'm'}(\theta,\phi)\sin\theta d\theta d\phi=\delta_{ll'}\delta_{mm'} \tag{3.8}$$

where one can show with (12.20) that

$$Y^*_{lm}(\theta,\phi)=(-1)^mY_{l,-m}(\theta,\phi) \tag{3.9}$$

It is useful to note that

$$\int_0^{2\pi}\int_0^{\pi}Y_{lm}(\theta,\phi)\sin\theta d\theta d\phi=\sqrt{4\pi}\delta_{l0}\delta_{m0} \tag{3.10}$$

and

$$Y_{l,0}(0,0)=\sqrt{\frac{2l+1}{4\pi}} \tag{3.11}$$

Fully normalized spherical harmonics are given by the routine `sphericalY`. The routine returns the values at points (θ,ϕ) up to degree L for all $\pm m$, linearly indexed. All the harmonics are returned because it is fastest to compute Legendre polynomials recursively and because field expansions often need all the harmonics. The output is a 2D array with first dimension with the points (θ,ϕ) and second dimension with harmonics size L^2+2L. This is to facilitate matrix-vector multiplication of the harmonics with a column vector of expansion coefficients. After such multiplication, the result has to be reshaped to the size of the input arrays. Use the string switch `'mono'` to include the monopole term, in which case the second dimension will be size L^2+2L+1. A naive computation of Y_{lm} would compute and divide the factorials directly. This is only accurate to about $L=21$. It is best to use the fully normalized Legendre polynomials, then only a factor of $1/\sqrt{2\pi}$ is needed to complete the normalization. Finally, this routine does not take advantage of the separability of θ and ϕ in the computation if, for example, the arrays contain redundant points on evenly spaced grids over the sphere.

```
function [ylm] = sphericalY(L,theta,phi,str)
% Fully normalized spherical harmonics
%
% Y_lm(theta,phi) = sqrt((2l+1)/4pi*(l-m)!/(l+m)!)
%                       *P_l^m(cos(theta)) e^(i m phi)
%
%                   (-1)^m included in P_l^m
%
% L:            Maximum harmonic degree L
```

```
% theta, phi:   Spherical angles in radians
% str:          [optional] 'mono' to include monopole term
%
% ylm:          Spherical harmonics evaluated at (theta,phi)
%               Dimensions: length(theta(:)) x N
%               N = L^2 + 2*L (N = L^2 + 2*L + 1 with monopole)
%
% Dependencies: ind2lm, Plm

N = L^2 + 2*L + 1;
[l, m] = ind2lm((1:N),'mono');
[Phi,M] = ndgrid(phi(:),m);
ylm = (1/sqrt(2*pi))*Plm(L,cos(theta(:))).'.*exp(1i*M.*Phi);
if nargin == 4
    if ~strcmp(str,'mono')
        error('bad string')
    end
    return
else
    ylm = ylm(:,2:end);
end
```

3.2.2 Angular Momentum Operators

The angular momentum operators, $\mathcal{L}$, for spherical harmonics are quite useful and will be used later to express vector spherical harmonics in Cartesian components. Several properties are repeated here from [11], which can also be found in textbooks (the operators are usually denoted with L, rather than $\mathcal{L}$).

$$\mathcal{L}^2 Y_{lm} = l(l+1)Y_{lm} \tag{3.12}$$

$$\mathcal{L}^2 = \mathcal{L}_x^2 + \mathcal{L}_y^2 + \mathcal{L}_z^2 \tag{3.13}$$

where

$$\begin{aligned} \mathcal{L} &= -i\mathbf{r} \times \nabla & (3.14)\\ &= \mathcal{L}_x\hat{x} + \mathcal{L}_y\hat{y} + \mathcal{L}_z\hat{z} & (3.15) \end{aligned}$$

where $\mathcal{L}_x$, $\mathcal{L}_y$, and $\mathcal{L}_z$ are differential operators that act on Cartesian vector fields. The components are combined to create raising and lowering operators:

$$\begin{aligned} \mathcal{L}_+ &= \mathcal{L}_x + i\mathcal{L}_y = e^{i\phi}\left(\frac{\partial}{\partial\theta} + i\cot\theta\frac{\partial}{\partial\phi}\right) & (3.16)\\ \mathcal{L}_- &= \mathcal{L}_x - i\mathcal{L}_y = e^{-i\phi}\left(-\frac{\partial}{\partial\theta} + i\cot\theta\frac{\partial}{\partial\phi}\right) & (3.17)\\ \mathcal{L}_z &= -i\frac{\partial}{\partial\phi} & (3.18) \end{aligned}$$

Adding and subtracting the raising and lower operators give

$$\begin{aligned} \mathcal{L}_x &= \frac{1}{2}\left(\mathcal{L}_+ + \mathcal{L}_-\right) & (3.19)\\ \mathcal{L}_y &= \frac{1}{2i}\left(\mathcal{L}_+ - \mathcal{L}_-\right) & (3.20) \end{aligned}$$

The operators have the property that

$$\begin{aligned} \mathcal{L}_+ Y_{lm} &= \sqrt{(l-m)(l+m+1)}Y_{l,m+1} & (3.21)\\ \mathcal{L}_- Y_{lm} &= \sqrt{(l+m)(l-m+1)}Y_{l,m-1} & (3.22)\\ \mathcal{L}_z Y_{lm} &= mY_{l,m} & (3.23) \end{aligned}$$

3.3 Scalar Spherical Wave Functions

The spherical scalar wave functions are solutions to the Helmholtz wave equation in free space in spherical coordinates. They are a product of spherical Bessel functions and spherical harmonics. The form of the Bessel function determines whether the wave is incoming or outgoing, specifically whether the field is regular at the origin or radiating, though they are all standing waves:

$$\text{Incoming regular wave:} \quad Rg\psi_{lm}(k,\mathbf{r}) = j_l(kr)Y_{lm}(\theta,\phi) \tag{3.24}$$

$$\text{Outgoing radiating wave:} \quad \psi_{lm}(k,\mathbf{r}) = h_l^{(1)}(kr)Y_{lm}(\theta,\phi) \tag{3.25}$$

where $j_l(kr)$ and $h_l^{(1)}(kr)$ are spherical Bessel and Hankel functions, respectively, k is the background wavenumber, and Rg means being regular (non-singular) at the origin which uses $j_l(kr)$. The wave functions form a complete set, so any scalar field can be expanded as an infinite sum:

$$\text{Incoming regular field:} \quad \phi(\mathbf{r}) = \sum_{l=0}^{\infty}\sum_{m=-l}^{l} a_{lm}Rg\psi_{lm}(k,\mathbf{r}) \tag{3.26}$$

$$\text{Outgoing radiating field:} \quad \phi(\mathbf{r}) = \sum_{l=0}^{\infty}\sum_{m=-l}^{l} b_{lm}\psi_{lm}(k,\mathbf{r}) \tag{3.27}$$

The far-field scalar wave functions for radiating waves are found by taking the large argument limit of the Hankel function

$$\lim_{kr\to\infty}\psi_{lm}(k,\mathbf{r}) = \frac{e^{ikr}}{kr}i^{-l-1}Y_{lm}(\theta,\phi) \tag{3.28}$$

The addition theorem for the scalar Green's function is given in terms of regular and radiating waves as, [2],

$$g(\mathbf{r},\mathbf{r}') = ik\sum_{l=0}^{\infty}\sum_{m=-l}^{l} Rg\psi_{lm}(k,\mathbf{r}_<)\psi_{lm}(k,\mathbf{r}_>) \tag{3.29}$$

where $\mathbf{r}_<$ is for the smaller of $|\mathbf{r}|$ and $|\mathbf{r}'|$ and $\mathbf{r}_>$ is for the greater of $|\mathbf{r}|$ and $|\mathbf{r}'|$.

The routine `psilm` returns $\psi_{lm}(k,\mathbf{r})$ at points (r,θ,ϕ) up to maximum degree L, for all $\pm m$, linearly indexed. k is real or complex. Like `sphericalY`, the first dimension has evaluation points and second dimension is size L^2+2L+1. Use the string switch `'rg'` for $Rg\psi_{lm}(k,\mathbf{r})$. Evaluation of the sum over harmonics for all points can be accomplished with matrix-vector multiplication then reshaping the result to match the dimensions of the input coordinates.

```
function [psi] = psilm(L,k,r,theta,phi,str)
% Scalar wave function:
%
%       psi_{lm}(kr) = h_l^(1)(kr)Y_l^m(theta,phi)
% or
%       Rg psi_{lm}(kr) = j_l(kr)Y_l^m(theta,phi)
%
% L:                Maximum harmonic degree L
% k:                Background wavenumber (real or complex)
% r,theta, phi:     Spherical coordinates, angles in radians
% str:              [optional] 'rg' for regular waves
%
% psilm:            Scalar wave function evaluated at (r,theta,phi)
%                   Dimensions: length(r(:)) x N
%                   N = L^2 + 2*L + 1
%
% Dependencies: lmtable, sphericalY, sbesselj, sbesselh

[l kr] = ndgrid(0:L,k*r(:));
if nargin == 6
    if ~strcmp(str,'rg')
        error('bad string')
    end
```

```
    bess = sbesselj(l,kr).';
else
    bess = sbesselh(l,kr).';
end
tab = lmtable(L,'mono');
psi = bess(:,tab(:,1)+1).*sphericalY(L,theta,phi,'mono');
```

3.4 Scalar Plane Wave Expansion

The expansion of scalar plane waves in terms of spherical waves at the origin is, [10],

$$e^{i\mathbf{k}\cdot\mathbf{r}} = 4\pi \sum_{l=0}^{\infty} \sum_{m=-l}^{l} i^l j_l(kr) Y_{lm}(\theta,\phi) Y_{lm}^*(\theta_k,\phi_k) \tag{3.30}$$

where the spherical harmonics are fully-normalized with Condon-Shortley phase and

$$\mathbf{r} = x\hat{x} + y\hat{y} + z\hat{z} \tag{3.31}$$

$$\mathbf{k} = k\hat{k} \tag{3.32}$$

$$\hat{k} = \sin\theta_k \cos\phi_k \hat{x} + \sin\theta_k \sin\phi_k \hat{y} + \cos\theta_k \hat{z} \tag{3.33}$$

The plane wave expansion coefficients are found by separating the regular scalar wave functions from (3.30) and collecting the remaining terms as expansion coefficients as

$$e^{i\mathbf{k}\cdot\mathbf{r}} = \sum_{l=0}^{\infty} \sum_{m=-l}^{l} a_{lm} Rg\psi_{lm}(k,\mathbf{r}) \tag{3.34}$$

$$a_{lm} = 4\pi i^l Y_{lm}^*(\theta_k,\phi_k) \tag{3.35}$$

The routine `scalarPlaneWaveCoef` routine takes as input the maximum harmonic degree L, and the spherical coordinates of the plane wave propagation direction, (θ_k, ϕ_k), and returns the a_{lm} scalar plane wave expansion coefficients for $l = 0, ..., L$, for all $\pm m$, linearly indexed.

```
function [alm] = scalarPlaneWaveCoef(L,thetak,phik)
% Scalar plane wave coefficients for fully normalized scalar
% wave functions
%
%   a_{lm} = 4\pi (i)^l Y_{l,m}^*(\theta_k,\phi_k)
%
% L:        maximum harmonic l = 0:L, m = -l:l
% thetak:   plane wave direction in theta (radians)
% phik:     plane wave direction in phi (radians)
%
% alm:      scalar plane wave coefficients linearly indexed
%
% Dependencies: lmtable, sphericalY

ylm = sphericalY(L,thetak,phik,'mono');
tab = lmtable(L,'mono');
l = tab(:,1);
alm = (4*pi)*(1i).^l.*conj(ylm(:));
```

3.5 Vector Spherical Harmonics

Vector spherical harmonics capture the angular variation of a vector field in spherical coordinates and are used for far-field radiation patterns. Their close cousin is the radially independent vector wave function usually notated $\mathbf{X}_{lm}$, [12]. The three fully normalized vector spherical harmonics, which form an orthonormal basis, are given by, [2, 10],

$$\mathbf{P}_{lm}(\theta,\phi) = \hat{r}Y_{lm}(\theta,\phi) \tag{3.36}$$

$$\mathbf{B}_{lm}(\theta,\phi) = \frac{1}{\sqrt{l(l+1)}} r\nabla Y_{lm}(\theta,\phi) \tag{3.37}$$

$$= N_{lm}\left[\hat{\theta}\frac{d}{d\theta}P_l^m(\cos\theta) + \hat{\phi}\frac{im}{\sin\theta}P_l^m(\cos\theta)\right]e^{im\phi} \tag{3.38}$$

$$= \frac{1}{\sqrt{2\pi l(l+1)}}\left[\hat{\theta}\frac{d}{d\theta}\widetilde{P}_l^m(\cos\theta) + \hat{\phi}\frac{im}{\sin\theta}\widetilde{P}_l^m(\cos\theta)\right]e^{im\phi} \tag{3.39}$$

$$\mathbf{C}_{lm}(\theta,\phi) = \frac{1}{\sqrt{l(l+1)}}\nabla\times[\mathbf{r}Y_{lm}(\theta,\phi)] \tag{3.40}$$

$$= N_{lm}\left[\hat{\theta}\frac{im}{\sin\theta}P_l^m(\cos\theta) - \hat{\phi}\frac{d}{d\theta}P_l^m(\cos\theta)\right]e^{im\phi} \tag{3.41}$$

$$= \frac{1}{\sqrt{2\pi l(l+1)}}\left[\hat{\theta}\frac{im}{\sin\theta}\widetilde{P}_l^m(\cos\theta) - \hat{\phi}\frac{d}{d\theta}\widetilde{P}_l^m(\cos\theta)\right]e^{im\phi} \tag{3.42}$$

where

$$N_{lm} = \frac{1}{\sqrt{l(l+1)}}\sqrt{\frac{2l+1}{4\pi}\frac{(l-m)!}{(l+m)!}} \tag{3.43}$$

The Condon-Shortley phase, $(-1)^m$, is included in the definition of the associated Legendre polynomials. The following relation exists between $\mathbf{B}_{lm}$ and $\mathbf{C}_{lm}$:

$$\mathbf{B}_{lm}(\theta,\phi) = \hat{r}\times\mathbf{C}_{lm}(\theta,\phi) \tag{3.44}$$

$$\mathbf{C}_{lm}(\theta,\phi) = -\hat{r}\times\mathbf{B}_{lm}(\theta,\phi) \tag{3.45}$$

In other words, $\mathbf{B}_{lm}\cdot\hat{\theta} = -\mathbf{C}_{lm}\cdot\hat{\phi}$ and $\mathbf{B}_{lm}\cdot\hat{\phi} = -\mathbf{C}_{lm}\cdot\hat{\theta}$. The fully normalized vector spherical harmonics have orthonormality

$$\int_0^{2\pi}\int_0^{\pi}\left\{\begin{array}{c}\mathbf{P}_{lm}(\theta,\phi)\cdot\mathbf{P}^*_{lm}(\theta,\phi)\\ \mathbf{B}_{lm}(\theta,\phi)\cdot\mathbf{B}^*_{lm}(\theta,\phi)\\ \mathbf{C}_{lm}(\theta,\phi)\cdot\mathbf{C}^*_{lm}(\theta,\phi)\end{array}\right\}\sin\theta d\theta d\phi = \delta_{ll'}\delta_{mm'} \tag{3.46}$$

and

$$\int_0^{2\pi}\int_0^{\pi}\mathbf{B}_{lm}(\theta,\phi)\cdot\mathbf{C}^*_{lm}(\theta,\phi)\sin\theta d\theta d\phi = 0 \tag{3.47}$$

Any spherical vector field (i.e., polarized spherical far-field pattern) can be expanded as

$$\mathbf{F}(\theta,\phi) = \sum_{l=1}^{\infty}\sum_{m=-l}^{l} b_{lm}\mathbf{B}_{lm}(\theta,\phi) + c_{lm}\mathbf{C}_{lm}(\theta,\phi) \tag{3.48}$$

The vector spherical harmonics can be expressed in Cartesian components using the angular momentum operators. $\mathbf{C}_{lm}$ written with the angular momentum operator is

$$\mathbf{C}_{lm} = \frac{\nabla\times[\mathbf{r}Y_{lm}]}{\sqrt{l(l+1)}} = \frac{-\mathbf{r}\times\nabla Y_{lm}}{\sqrt{l(l+1)}} = \frac{-i\mathcal{L}Y_{lm}}{\sqrt{l(l+1)}} \tag{3.49}$$

which leads to

$$\mathbf{C}_{lm} = \frac{-i}{\sqrt{l(l+1)}}\left[\frac{1}{2}\left(\hat{x}-i\hat{y}\right)e_{lm}Y_{l,m+1} + \frac{1}{2}\left(\hat{x}+i\hat{y}\right)f_{lm}Y_{l,m-1} + mY_{l,m}\hat{z}\right] \tag{3.50}$$

where

$$e_{lm} = \sqrt{(l-m)(l+m+1)} \tag{3.51}$$
$$f_{lm} = \sqrt{(l+m)(l-m+1)} \tag{3.52}$$

For the special case $(\theta,\phi)=(0,0)$, only the $m=\pm 1$ harmonics survive, which leads to

$$\mathbf{C}_{l,\pm 1}(0,0) = \frac{-i}{2}\left(\hat{x}\mp i\hat{y}\right)\sqrt{\frac{2l+1}{4\pi}} \tag{3.53}$$
$$\mathbf{B}_{l,\pm 1}(0,0) = \frac{-i}{2}\left(\hat{y}\pm i\hat{x}\right)\sqrt{\frac{2l+1}{4\pi}} \tag{3.54}$$

The second special case $(\theta,\phi)=(\pi,0)$ has again only $m=\pm 1$ harmonics. Using $Y_{lm}(\pi,0)=(-1)^l Y_{lm}(0,0)$ we get

$$\mathbf{C}_{l,\pm 1}(\pi,0) = (-1)^l\mathbf{C}_{l,\pm 1}(0,0) \tag{3.55}$$
$$\mathbf{B}_{l,\pm 1}(\pi,0) = (-1)^{l+1}\mathbf{B}_{l,\pm 1}(0,0) \tag{3.56}$$

The routine `BC` returns the components of $\mathbf{B}_{lm}$ and $\mathbf{C}_{lm}$, for all harmonics up to degree L, for all $\pm m$, linearly indexed along the second dimension with spherical points along the first dimension. There is duplication in returning each vector component, but none in the computation. Use the string switch `'norm'` for fully normalized $(1/\sqrt{l(l+1)})$.

The helper function `BCmult` takes the outputs from `BC` and expansion coefficients b_{lm} and c_{lm} and returns the field components of (3.48), F_θ, F_ϕ. These arrays are columnized and require reshaping to make them the same size as the `BC` input (θ,ϕ) arrays.

```
function [Bth, Bphi, Cth, Cphi] = BC(L,theta,phi,normstr)
% Vector spherical harmonics, defaults to partial normalization
%
% B_lm(theta,phi) = N_lm [theta_hat d/dtheta P_l^m(cos(theta)) +
%                         phi_hat im/sin(theta) P_l^m(cos(theta))]e^(im phi)
%
% C_lm(theta,phi) = N_lm [theta_hat im/sin(theta) P_l^m(cos(theta)) -
%                         phi_hat d/dtheta P_l^m(cos(theta))]e^(im phi)
%
% N_lm = (-1)^m/sqrt(l(l+1))*sqrt((2l+1)/4pi*(l-m)!/(l+m)!)
%
% L:                    Maximum harmonic degree L
% theta, phi:           Spherical angles in radians, any size
% normstr:              [optional] 'norm' for 1/sqrt(l(l+1)) full normalization
%
% Bth,Bphi,Cth,Cphi:    Vector spherical harmonics components
%                       evaluated at (theta,phi)
%                       Dimensions: length(theta(:)) x N
%                       N = L^2 + 2*L
%
% Dependencies: ind2lm, Plmp2, mPlmsin

N = L^2 + 2*L;
theta = theta(:)';
phi = phi(:)';

% d/dtheta P_l^m
dP = Plmp2(L,theta).';
```

```
% m P_l^m/sin(theta)
mPsin = mPlmsin(L,theta).';

% remaining normalizaton and exponential
[l, m] = ind2lm((1:N)');
[Phi,M] = ndgrid(phi,m);
ex = 1./sqrt(2*pi).*exp(1i*M.*Phi);
if nargin == 4
    if ~strcmp(normstr,'norm')
        error('bad string')
    end
    [~,L] = ndgrid(phi,l);
    ex =  ex.*1./sqrt(L.*(L+1));
end
dP = dP.*ex;
mPsin = 1i*mPsin.*ex;

% B and C components theta_hat, phi_hat
Bth = dP;
Bphi = mPsin;
Cth = mPsin;
Cphi = -dP;


function [Fth, Fphi] = BCmult(Bth,Bphi,Cth,Cphi,blm,clm)
% Vector spherical function from vector spherical harmonics
%
% blm, clm:                 expansion coefficients
% Bth, Bphi, Cth, Cphi:     outputs from BC
%
% Fth, Fphi:                theta/phi field components

blm = blm(:);
clm = clm(:);
Fth = Bth*blm + Cth*clm;
Fphi = Bphi*blm + Cphi*clm;
```

3.6 Vector Spherical Wave Functions

The vector spherical wave functions are solutions to the free-space vector wave equation in spherical coordinates for electric or magnetic fields, [2, 10]. Analogous to the scalar wave functions, they are products of spherical Bessel functions and the vector spherical harmonics, and come in regular and radiating forms.

$$
\begin{aligned}
Rg\mathbf{M}_{lm}(k,\mathbf{r}) &= \frac{1}{\sqrt{l(l+1)}}\nabla\times[\mathbf{r}Rg\psi_{lm}(k,\mathbf{r})] && (3.57)\\
&= j_l(kr)\mathbf{C}_{lm}(\theta,\phi) && (3.58)\\
\mathbf{M}_{lm}(k,\mathbf{r}) &= \frac{1}{\sqrt{l(l+1)}}\nabla\times[\mathbf{r}\psi_{lm}(k,\mathbf{r})] && (3.59)\\
&= h_l^{(1)}(kr)\mathbf{C}_{lm}(\theta,\phi) && (3.60)\\
Rg\mathbf{N}_{lm}(k,\mathbf{r}) &= \frac{1}{k}\nabla\times Rg\mathbf{M}_{lm}(k,\mathbf{r}) && (3.61)\\
&= \sqrt{l(l+1)}\frac{j_l(kr)}{kr}\mathbf{P}_{lm}(\theta,\phi)+\frac{[krj_l(kr)]'}{kr}\mathbf{B}_{lm}(\theta,\phi) && (3.62)\\
\mathbf{N}_{lm}(k,\mathbf{r}) &= \frac{1}{k}\nabla\times\mathbf{M}_{lm}(k,\mathbf{r}) && (3.63)\\
&= \sqrt{l(l+1)}\frac{h_l^{(1)}(kr)}{kr}\mathbf{P}_{lm}(\theta,\phi)+\frac{\left[krh_l^{(1)}(kr)\right]'}{kr}\mathbf{B}_{lm}(\theta,\phi) && (3.64)
\end{aligned}
$$

Note that $\mathbf{M}_{lm}(k,\mathbf{r})$ contains only $\hat{\theta}$ and $\hat{\phi}$ components, while $\mathbf{N}_{lm}(k,\mathbf{r})$ contains the same plus an $\hat{r}$ component. $\mathbf{N}_{lm}(k,\mathbf{r})$ holds the near-field radial part of a field, which disappears in the far-field. Here, $\mathbf{P}_{lm}$, $\mathbf{B}_{lm}$ and $\mathbf{C}_{lm}$ are fully normalized. If partially normalized wave functions are desired, then both sides of the equations are multiplied by $\sqrt{l(l+1)}$.

The two wave functions are related by

$$
\begin{aligned}
\mathbf{M}_{lm}(k,\mathbf{r}) &= \frac{1}{k}\nabla\times\mathbf{N}_{lm}(k,\mathbf{r}) && (3.65)\\
\mathbf{N}_{lm}(k,\mathbf{r}) &= \frac{1}{k}\nabla\times\mathbf{M}_{lm}(k,\mathbf{r}) && (3.66)
\end{aligned}
$$

An electric or magnetic field can be written as a sum of vector wave functions. This is the multipole expansion of the vector field. $\mathbf{M}_{lm}(k,\mathbf{r})$ and $\mathbf{N}_{lm}(k,\mathbf{r})$ are linearly independent, so any field is a linear combination of both

$$
\mathbf{E}(\mathbf{r})=\sum_{l=1}^{\infty}\sum_{m=-l}^{l}a_{lm}\mathbf{M}_{lm}(k,\mathbf{r})+b_{lm}\mathbf{N}_{lm}(k,\mathbf{r}) \tag{3.67}
$$

$$
\mathbf{H}(\mathbf{r})=\frac{k}{i\omega\mu}\sum_{l=1}^{\infty}\sum_{m=-l}^{l}a_{lm}\mathbf{N}_{lm}(k,\mathbf{r})+b_{lm}\mathbf{M}_{lm}(k,\mathbf{r}) \tag{3.68}
$$

where a_{lm} and b_{lm} are complex expansion coefficients. These equations are for radiating fields. Expressions for incoming fields will use the Rg counterparts.

The addition theorem for the dyadic Green's function can be written as an outer product of normalized vector wave functions, [2],

$$
\overline{\mathbf{G}}(\mathbf{r},\mathbf{r}')=ik\sum_{l=1}^{\infty}\sum_{m=-l}^{l}Rg\mathbf{M}_{lm}(k,\mathbf{r})\hat{\mathbf{M}}_{lm}(k,\mathbf{r}')+Rg\mathbf{N}_{lm}(k,\mathbf{r})\hat{\mathbf{N}}_{lm}(k,\mathbf{r}') \tag{3.69}
$$

where $\hat{}$ means conjugation of the angular function only. This equation applies under the condition $|\mathbf{r}| < |\mathbf{r}'|$. When $|\mathbf{r}| > |\mathbf{r}'|$ the equation changes so that the Rg is instead applied to $\hat{\mathbf{M}}_{lm}(k,\mathbf{r})$ and $\hat{\mathbf{N}}_{lm}(k,\mathbf{r})$. The far-field vector wave functions are found by taking the large argument limit of the Bessel functions in $\mathbf{M}_{lm}(k,\mathbf{r})$ and $\mathbf{N}_{lm}(k,\mathbf{r})$

$$\lim_{kr\to\infty} \mathbf{M}_{lm}(k,\mathbf{r}) = \frac{e^{ikr}}{kr} i^{-l-1} \mathbf{C}_{lm}(\theta,\phi) \tag{3.70}$$

$$\lim_{kr\to\infty} \mathbf{N}_{lm}(k,\mathbf{r}) = \frac{e^{ikr}}{kr} i^{-l} \mathbf{B}_{lm}(\theta,\phi) \tag{3.71}$$

Vector wave functions are orthogonal over the unit sphere as

$$\int \mathbf{M}_{lm}(k,\mathbf{r}) \cdot \hat{\mathbf{M}}_{l'm'}(k,\mathbf{r}) d\Omega = \left(h_l^{(1)}(kr)\right)^2 \delta_{ll'}\delta_{mm'} \tag{3.72}$$

$$\int \mathbf{N}_{lm}(k,\mathbf{r}) \cdot \hat{\mathbf{N}}_{l'm'}(k,\mathbf{r}) d\Omega = \frac{1}{(kr)^2}\left[l(l+1)\left(h_l^{(1)}(kr)\right)^2 + \left(\left[krh_l^{(1)}(kr)\right]'\right)^2\right]\delta_{ll'}\delta_{mm'} \tag{3.73}$$

$$\int \mathbf{M}_{lm}(k,\mathbf{r}) \cdot \hat{\mathbf{N}}_{l'm'}(k,\mathbf{r}) d\Omega = 0 \tag{3.74}$$

The cross products dotted with the radial vector and integrated over the sphere are

$$\int \left(\mathbf{M}_{lm}(k,\mathbf{r}) \times \hat{\mathbf{M}}_{l'm'}(k,\mathbf{r})\right) \cdot \hat{\mathbf{r}} d\Omega = 0 \tag{3.75}$$

$$\int \left(\mathbf{N}_{lm}(k,\mathbf{r}) \times \hat{\mathbf{N}}_{l'm'}(k,\mathbf{r})\right) \cdot \hat{\mathbf{r}} d\Omega = 0 \tag{3.76}$$

$$\int \left(\mathbf{M}_{lm}(k,\mathbf{r}) \times \hat{\mathbf{N}}_{l'm'}(k,\mathbf{r})\right) \cdot \hat{\mathbf{r}} d\Omega = h_l^{(1)}(kr)\frac{\left[krh_l^{(1)}\right]'}{kr}\delta_{ll'}\delta_{mm'} \tag{3.77}$$

These work for any combination of regular or radiating wave functions, simply substitute the correct Bessel functions.

The routine `MN` returns the components of $\mathbf{M}_{lm}(k,\mathbf{r})$ and $\mathbf{N}_{lm}(k,\mathbf{r})$, for all harmonics up to degree L, for all $\pm m$, linearly indexed. This routine calls `BC`. Like `BC`, the outputs are 2D arrays with L^2+2L harmonics along the second dimension and the evaluation points at (r,θ,ϕ) along the first. Use the string switch `'rg'` for regular waves, `'hat'` for angular conjugation, and `'norm'` to use fully normalized vector spherical harmonics. It defaults to partial normalization. For regular waves, the spherical Bessel function routines from Chapter 12.2 take care of kr near the origin. Use the following input constructions for `MN`:

Table 3.3: `MN` Input Options

$\mathbf{M}_{lm}(k,\mathbf{r})$, $\mathbf{N}_{lm}(k,\mathbf{r})$	`MN(L,k,r,theta,phi)`
$Rg\mathbf{M}_{lm}(k,\mathbf{r})$, $Rg\mathbf{N}_{lm}(k,\mathbf{r})$	`MN(L,k,r,theta,phi,'rg')`
$\hat{\mathbf{M}}_{lm}(k,\mathbf{r})$, $\hat{\mathbf{N}}_{lm}(k,\mathbf{r})$	`MN(L,k,r,theta,phi,[],'hat')`
$Rg\hat{\mathbf{M}}_{lm}(k,\mathbf{r})$, $Rg\hat{\mathbf{N}}_{lm}(k,\mathbf{r})$	`MN(L,k,r,theta,phi,'rg','hat')`
Fully normalized $\mathbf{B}_{lm}$ and $\mathbf{C}_{lm}$	`MN(L,k,r,theta,phi,...,...,'norm')`

The helper function `MNmult` takes the outputs from `MN` and expansion coefficients a_{lm} and b_{lm} and returns the electric field components E_r, E_θ, E_ϕ. These arrays are columnized and require reshaping to make them the same size as the `MN` input (r,θ,ϕ) arrays. Magnetic field components are obtained by swapping a_{lm} and b_{lm} and multiplying by $k/i\omega\mu$ as in (3.68).

```
function [Mth Mphi Nr Nth Nphi] = MN(L,k,r,theta,phi,rgstr,hatstr,normstr)
% Vector wave functions:
%
%       M_lm(kr) = h_l^(1)(kr)C_lm(theta,phi)
% and
%       N_lm(kr) = l(l+1)h_l^(1)(kr)/kr P_lm(theta,phi)
%                     + [kr h_l^(1)(kr)]'/kr B_lm(theta,phi)
% or
%
%       RgM_lm(kr) = j_l(kr)C_lm(theta,phi)
% and
%       RgN_lm(kr) = l(l+1)j_l(kr)/kr P_lm(theta,phi)
%                       + [kr j_l(kr)]'/kr B_lm(theta,phi)
%
% L:                Maximum harmonic degree L
% k:                Background wavenumber (real or complex)
% r,theta,phi:      Spherical coordinates, angles in radians
% str:              'rg' for regular waves
% hat:              'hat' for angular conjugation: \hatM and \hatN
% normstr:          'norm' for full, 1/sqrt(1(l+1)), normalization
%
% Mth,Mphi,         Vector wave function evaluated at (r,theta,phi)
% Nr,Nth,Nphi:      Dimensions: lenght(r(:)) x N
%                   N = L^2 + 2*L
%
% Dependencies: ind2lm, sphericalY, sbesselj, sbesselj2,
%               sbesseljp, sbesselh, sbesselhp, BC

N = L^2 + 2*L;
[l m] = ind2lm((1:N)');
kr = k*r(:);
[l2 KR] = ndgrid(1:L,kr);
if nargin < 5
    error('not enough inputs')
end
if nargin >= 6 && strcmp(rgstr,'rg')
    bess1 = sbesselj(l2,KR).';      % j_l(kr), handles lim r->0
    bess2 = sbesselj2(l2,KR).';     % j_l(kr)/kr, "
    bess3 = sbesseljp(l2,KR).';     % j'_l(kr), "
else
    if ~isempty(rgstr)
        error('bad string')
    end
    bess1 = sbesselh(l2,KR).';          % h_l^(1)(kr)
    bess2 = (sbesselh(l2,KR)./KR).';    % h_l^(1)(kr)/kr
    bess3 = sbesselhp(l2,KR).';         % h'_l^(1)(kr)
end
bess4 = bess2 + bess3;
if nargin == 8 && strcmp(normstr,'norm')
    ylm = repmat(sqrt(l.*(l+1))',length(kr),1).*sphericalY(L,theta,phi);
    [Bth, Bphi, Cth, Cphi] = BC(L,theta,phi,'norm');
elseif nargin == 8 && ~isempty(normstr)
    error('bad string')
else
    ylm = repmat((l.*(l+1))',length(kr),1).*sphericalY(L,theta,phi);
    [Bth, Bphi, Cth, Cphi] = BC(L,theta,phi);
end
if nargin >= 7 && strcmp(hatstr,'hat')
    ylm = conj(ylm);
    Bth = conj(Bth);
    Bphi = conj(Bphi);
    Cth = conj(Cth);
    Cphi = conj(Cphi);
else
    if ~isempty(hatstr),
        error('bad string')
    end
end
```

```
Mth = bess1(:,l).*Cth;
Mphi = bess1(:,l).*Cphi;
Nr = bess2(:,l).*ylm;
Nth = bess4(:,l).*Bth;
Nphi  = bess4(:,l).*Bphi;
```

```
function [Er Eth Ephi] = MNmult(Mth,Mphi,Nr,Nth,Nphi,alm,blm)
% Spherical electric field from vector spherical wave functions
%
% alm, blm:                 expansion coefficients
% Mth,Mphi,Nr,Nth,Nphi:     outputs from MN
%
% Er,Eth,Ephi:              r/theta/phi field components

alm = alm(:);
blm = blm(:);
Er = Nr*blm;
Eth = Mth*alm + Nth*blm;
Ephi = Mphi*alm + Nphi*blm;
```

3.7 Vector Plane Wave Expansion

Here we derive the vector wave function expansion coefficients for vector plane waves with arbitrary propagation direction. These are needed when combining plane wave excitations and vector wave functions, for instance, in T-matrix scattering problems.

3.7.1 Plane Wave Representation

Let the electric and magnetic fields of the vector plane wave be written

$$\mathbf{E}(\mathbf{r}) = \mathbf{E}e^{i\mathbf{k}\cdot\mathbf{r}} \tag{3.78}$$

$$\mathbf{H}(\mathbf{r}) = \mathbf{H}e^{i\mathbf{k}\cdot\mathbf{r}} \tag{3.79}$$

where

$$\mathbf{r} = x\hat{x} + y\hat{y} + z\hat{z} \tag{3.80}$$

$$\mathbf{k} = k\hat{k} \tag{3.81}$$

$$\hat{k} = \sin\theta_k \cos\phi_k \hat{x} + \sin\theta_k \sin\phi_k \hat{y} + \cos\theta_k \hat{z} \tag{3.82}$$

and

$$\mathbf{E} = E_x\hat{x} + E_y\hat{y} + E_z\hat{z} \tag{3.83}$$

$$\mathbf{H} = \frac{1}{\eta}\hat{k} \times \mathbf{E} \tag{3.84}$$

$$= H_x\hat{x} + H_y\hat{y} + H_z\hat{z} \tag{3.85}$$

where η is the characteristic impedance of free-space.

3.7.2 Vector Plane Wave Coefficients

To solve for the expansion coefficients about the origin, write the electric and magnetic fields as an expansion of regular waves at the origin

$$\mathbf{E}(\mathbf{r}) = \sum_{l=1}^{\infty}\sum_{m=-l}^{l} a_{lm} Rg\mathbf{M}_{lm}(k,\mathbf{r}) + b_{lm} Rg\mathbf{N}_{lm}(k,\mathbf{r}) \tag{3.86}$$

$$\mathbf{H}(\mathbf{r}) = \frac{k}{i\omega\mu}\sum_{l=1}^{\infty}\sum_{m=-l}^{l} a_{lm} Rg\mathbf{N}_{lm}(k,\mathbf{r}) + b_{lm} Rg\mathbf{M}_{lm}(k,\mathbf{r}) \tag{3.87}$$

Multiplying both fields by $Rg\hat{\mathbf{M}}_{lm}(k,\mathbf{r})$, integrating over the sphere, applying orthogonality for fully normalized wave functions, (3.72), and canceling one factor of the Bessel functions, we get

$$a_{lm}j_l(kr) = \int \mathbf{C}^*_{lm}(\theta,\phi)\cdot\mathbf{E}(\mathbf{r})d\Omega \tag{3.88}$$

$$b_{lm}j_l(kr)\frac{k}{i\omega\mu} = \int \mathbf{C}^*_{lm}(\theta,\phi)\cdot\mathbf{H}(\mathbf{r})d\Omega \tag{3.89}$$

Substituting (3.78) into (3.88), then substituting the scalar plane wave expansion, (3.30), which cancels the remaining Bessel function, and exchanging the sum and integration

$$a_{lm} = 4\pi\mathbf{E}\cdot\sum_{l'=0}^{\infty}\sum_{m'=-l'}^{l'} i^l\int \mathbf{C}^*_{lm}(\theta,\phi)Y_{l'm'}(\theta,\phi)Y^*_{l'm'}(\theta_k,\phi_k)d\Omega \tag{3.90}$$

Using (3.50) it can be shown that the integral and sum reduces for both coefficients to

$$a_{lm} = 4\pi i^l\mathbf{C}^*_{lm}(\theta_k,\phi_k)\cdot\mathbf{E} \tag{3.91}$$

$$b_{lm} = 4\pi i^{l+1}\mathbf{C}^*_{lm}(\theta_k,\phi_k)\cdot(\eta\mathbf{H}) \tag{3.92}$$

$$= -4\pi i^{l+1}\mathbf{B}^*_{lm}(\theta_k,\phi_k)\cdot\mathbf{E} \tag{3.93}$$

where the last equation comes from taking $\hat{k}=\hat{r}$ in (3.84). Taking the dot product between (3.50) and (3.83) the coefficients are written out as

$$a_{lm} = \frac{4\pi i^{l+1}}{\sqrt{l(l+1)}}\left[\frac{1}{2}(E_x+iE_y)\sqrt{(l-m)(l+m+1)}Y^*_{l,m+1}(\theta_k,\phi_k)\right.$$
$$\left.+\frac{1}{2}(E_x-iE_y)\sqrt{(l+m)(l-m+1)}Y^*_{l,m-1}(\theta_k,\phi_k)+E_z mY^*_{lm}(\theta_k,\phi_k)\right] \tag{3.94}$$

$$b_{lm} = ia_{lm}(\mathbf{E}\rightarrow\eta\mathbf{H}=\hat{k}\times\mathbf{E}) \tag{3.95}$$

The plane wave expansion coefficients have left and right circularly polarized amplitudes embedded in them.

The routine `vectorPlaneWaveCoef` routine takes as input the maximum harmonic degree L, the electric field components of the plane wave, E_x, E_y, and E_z, and the spherical coordinates of the plane wave propagation direction, (θ_k,ϕ_k), and returns the a_{lm} and b_{lm} vector plane wave expansion coefficients for $l=1,...,L$, for all $\pm m$, linearly indexed.

```
function [alm, blm] = vectorPlaneWaveCoef(L,Ex,Ey,Ez,thetak,phik)
% Multipole coefficients for plane wave at the origin
% for fully-normalized vector wave functions.
%
% L:                Maximum harmonic L
% Ex,Ey,Ez:         Electric field components
% thetak, phik:     k-vector direction (radians)
%
% alm, blm:         plane wave multipole coefficients,
%                   harmonics 1:L, all m, linearly indexed
%
% Dependencies: sphericalY

tot = L^2 + 2*L;
alm = zeros(tot,1);
blm = zeros(tot,1);
khat = [sin(thetak)*cos(phik) sin(thetak)*sin(phik) cos(thetak)];
H = cross(khat,[Ex,Ey,Ez]);
Hx = H(1);
Hy = H(2);
Hz = H(3);
ylm = conj(sphericalY(L,thetak,phik));
e1 = 0.5*(Ex + 1i*Ey);
```

```
e2 = 0.5*(Ex - 1i*Ey);
h1 = 0.5*(Hx + 1i*Hy);
h2 = 0.5*(Hx - 1i*Hy);
for l=1:L,
    c1 = 4*pi*1i^(l+1)/sqrt(l*(l+1));
    for m=-l:l,
        elm = sqrt((l-m)*(l+m+1));
        flm = sqrt((l+m)*(l-m+1));
        ind = l^2 + l + m;
        atmp = Ez*m*ylm(ind);
        btmp = Hz*m*ylm(ind);
        if m < l
            atmp = atmp + e1*elm*ylm(ind+1);
            btmp = btmp + h1*elm*ylm(ind+1);
        end
        if m > -l
            atmp = atmp + e2*flm*ylm(ind-1);
            btmp = btmp + h2*flm*ylm(ind-1);
        end
        alm(ind) = c1*atmp;
        blm(ind) = 1i*c1*btmp;
    end
end
```

3.7.3 $\hat{z}$-Propagating Plane Wave

Here we derive the special case of a vector plane wave propagating in the $\hat{z}$ direction, $\theta_k = \phi_k = 0$, and $E_z = 0$. This is useful for radar backscatter computations. Only the $m = \pm 1$ harmonics are needed, and the expansion coefficients become

$$a_{l,\pm 1} = 2\pi i^{l+1} \left(E_x \mp iE_y\right) Y_{l,\pm 1}(0,0) \tag{3.96}$$

$$b_{l,\pm 1} = -2\pi i^{l} \eta \left(H_x \mp iH_y\right) Y_{l,\pm 1}(0,0) \tag{3.97}$$

Using (3.11) and the fact that $H_x = -E_y/\eta$ and $H_y = E_x/\eta$ for a $\hat{z}$-propagating plane wave, one can show

$$a_{l,\pm 1} = \sqrt{\pi(2l+1)} i^{l+1} \left(E_x \mp iE_y\right) \tag{3.98}$$

$$b_{l,\pm 1} = \pm a_{l,\pm 1} \tag{3.99}$$

The routine `vectorPlaneWaveCoefZ` works like `vectorPlaneWaveCoef`. It takes as input the maximum harmonic degree L, and the electric field components of the $\hat{z}$-propagating plane wave, E_x, E_y, and returns the a_{lm} and b_{lm} vector plane wave expansion coefficients for $l = 1, ..., L$, for all $\pm m$, linearly indexed.

```
function [alm, blm] = vectorPlaneWaveCoefZ(L,Ex,Ey)
% Multipole coefficients for z-propagating plane wave at the orgin
% for fully-normalized vector wave functions.
%
% L:            Maximum harmonic L
% Ex,Ey:        Electric field components
%
% alm, blm:     z-hat plane wave multipole coefficients,
%               harmonics 1:L, all m, linearly indexed

tot = L^2 + 2*L;
alm = zeros(tot,1);
blm = zeros(tot,1);
for l=1:L,
    tmp = (1i)^(l+1)*sqrt(pi*(2*l+1));
    ind = l^2 + l + 1;
    alm(ind) = tmp*(Ex - 1i*Ey);
    blm(ind) = alm(ind);
    ind = l^2 + l - 1;
    alm(ind) = tmp*(Ex + 1i*Ey);
    blm(ind) = -alm(ind);
end
```

3.8 Band-limited Fields and Required Number of Spherical Harmonics

The number of spherical harmonics, or spherical spectral components, that are needed to fully capture the information contained in a far-field radiation pattern depends only on the size of the object and the background wavenumber. This is true regardless of the make up of the object and applies to both scalar and vector fields. For instance, the radiation pattern could be the scattered field from a natural object or the field radiated by an antenna array.

It is commonly stated that $O(kd)$ spherical harmonics are required to represent a spherical field pattern, where k is the background wavenumber and d is the diameter of the smallest sphere that encloses the object. $O(kd)$ actually gives the maximum degree harmonic L, where all m need to be included in the expansion. Therefore, a total of $N = L^2 + 2L + 1$ harmonics are needed in the case of scalar waves. This is also equal to the number of angular points needed to Nyquist-sample the spatial pattern of a field over the sphere. This criteria has been studied in detail, [13], and a more precise value for the maximum degree harmonic, written in terms of the radius of the smallest enclosing sphere, a, is

$$L \approx \left\lceil 1.1ka \left(1 + \frac{1}{ka}\right) \right\rceil \tag{3.100}$$

This is a remarkable fact about fields. No matter the make up of a scattering object or antenna, so long as it is contained within a radius a relative to the origin, the angular variation of the far-field can only oscillate as fast as the maximum harmonic $(l, m) = (L, \pm L)$. This is what is meant by the phrase band-limited field, because the spherical harmonic spectrum of any radiated field is finite.

A quick derivation and visual of this idea follows. Start with the volume integral equation for the scalar scattered field

$$\phi_{sca}(\mathbf{r}) = \int g(\mathbf{r}, \mathbf{r}')s(\mathbf{r}')dV' \tag{3.101}$$

where $s(\mathbf{r})$ is a volume source. Substitute the addition theorem for the scalar Green's function, (3.29), so that (3.101) is expanded

$$\phi_{sca}(\mathbf{r}) = \sum_{lm} a_{lm}\psi_{lm}(\mathbf{r}) \tag{3.102}$$

$$a_{lm} = ik \int Rg\psi_{lm}(\mathbf{r}')s(\mathbf{r}')dV' \tag{3.103}$$

Choose a point source in spherical coordinates that has unit amplitude at radius a along the z axis:

$$s(\mathbf{r}) = \frac{1}{r^2 \sin\theta}\delta(r - a)\delta(\theta)\delta(\phi) \tag{3.104}$$

Evaluating (3.103) with this source over spherical coordinates gives the expansion coefficients

$$a_{lm} = ikj_l(ka)Y_{lm}(0, 0) \tag{3.105}$$

$$= ik\sqrt{\frac{2l+1}{4\pi}}j_l(ka) \tag{3.106}$$

The power in each harmonic is given by the magnitude squared of the coefficients

$$|a_{lm}|^2 = k^2\frac{(2l+1)}{4\pi}|j_l(ka)|^2 \tag{3.107}$$

Finally, we can normalize this to capture only the dependence on l

$$|\widetilde{a}_{lm}|^2 = (2l+1)|j_l(ka)|^2 \tag{3.108}$$

Figure 3.8 left shows the magnitude of the normalized coefficients as a function of ka and l. There is very quickly little harmonic content in the expansion coefficients above the line $l = ka$. Figure 3.8 right shows the normalized harmonics for $ka = 25$ as well as the cutoff predicted by (3.100).

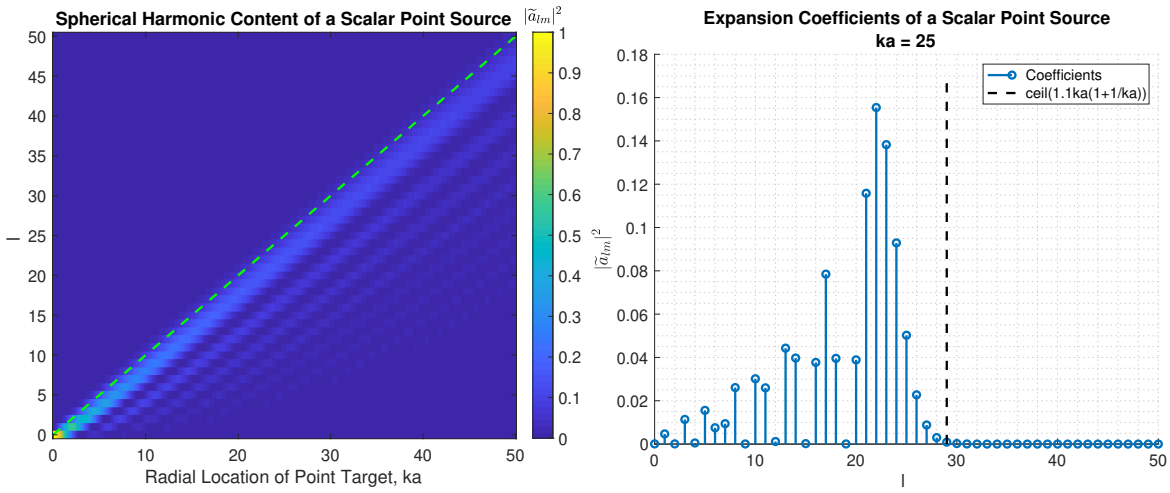


Figure 3.1: Spherical harmonic content of a scalar point source at $z = a$. Left: coefficients vs ka with the line $l = ka$ plotted. Right: example of $ka = 25$ with (3.100) plotted.

Chapter 4

Rotation

This chapter gives routines for computing rotation matrices that operate on the expansion coefficients of spherical harmonics and spherical wave functions in order to rotate fields or view fields from a rotated frame. The rotation matrices work the same for any of the wave functions in Chapter 3.

4.1 Euler Rotation

3D rotation of a Cartesian reference frame can be accomplished with three Euler angles (α, β, γ), where the sequence rotates the frame successively about one of its axes. The angles can take any value of radians. The most common rotation sequence is denoted Z-X'-Z", sometimes just called ZXZ. This consists of a right-handed rotation of α about the z axis, followed by a right-handed rotation of β about the x axis of the rotated frame (X'), followed by a right-handed rotation of γ about the z axis of the rotated frame (Z"). This is represented in a rotation matrix as:

$$\mathbf{R}_{ZX'Z''}(\alpha,\beta,\gamma) = \mathbf{R}_Z(\alpha)\mathbf{R}_X(\beta)\mathbf{R}_Z(\gamma) \tag{4.1}$$

$$= \begin{bmatrix} \cos(\alpha) & -\sin(\alpha) & 0 \\ \sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\beta) & -\sin(\beta) \\ 0 & \sin(\beta) & \cos(\beta) \end{bmatrix} \begin{bmatrix} \cos(\gamma) & -\sin(\gamma) & 0 \\ \sin(\gamma) & \cos(\gamma) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{4.2}$$

Each matrix is a right-handed rotation about the axes of the unrotated, or fixed, global frame. The reason the matrices are applied in what seems like reverse order, and applied about the axes of the fixed frame, is because each matrix stacks the rotations in front of it to create the effect that each rotation was applied about the axes of the rotated frame in the order (α, β, γ).

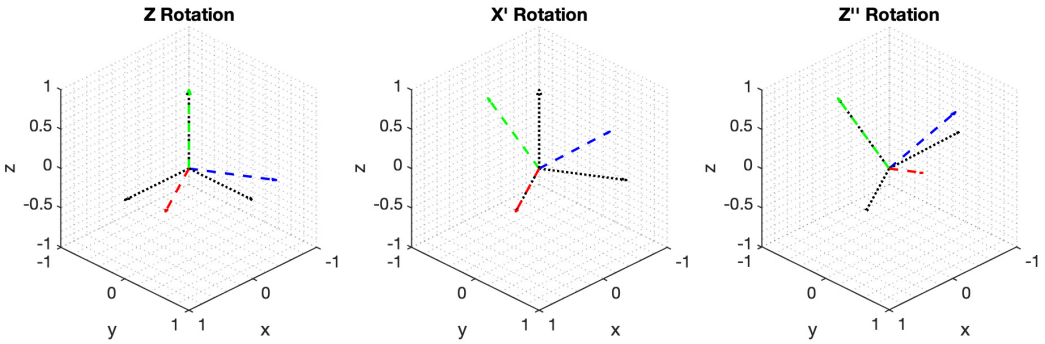


Figure 4.1: ZX'Z" rotation sequence $(\alpha, \beta, \gamma) = (\pi/6, \pi/5, \pi/4)$. First pane shows the rotation of α about the z axis from the original Cartesian frame (black dotted) to the new frame (red/blue/green). Second pane shows the second rotation of β about the x axis of the previously rotated frame. Third pane shows the final rotation of γ about the z axis of the previously rotated frame.

A point $\mathbf{x} = [x, y, z]^t$ that in the global frame is rotated to the point $\mathbf{x}' = [x', y', z']^t$ as

$$\mathbf{x}' = \mathbf{R}\mathbf{x} \tag{4.3}$$

In other words, the point is rotated through the global frame as though you are standing in the global frame. We call this a forward rotation. $\mathbf{R}$ is unitary, so the inverse is equal to the transpose, and

$$\mathbf{x} = \mathbf{R}^t\mathbf{x}' \tag{4.4}$$

The inverse rotation brings the point back, and applies the Euler angles in the reverse order and reverse direction. Given a forward set of angles (α, β, γ), applying the inverse is equivalent to fixing a point in the global frame and seeing it as you ride along with the rotating frame. We call this the inverse rotation.

Determining the ZXZ Euler angles from a rotation matrix is done with

$$\alpha = \arctan \frac{R_{13}}{-R_{23}} \tag{4.5}$$

$$\beta = \arctan \frac{\sqrt{R_{13}^2 + R_{23}^2}}{R_{33}} \tag{4.6}$$

$$\gamma = \arctan \frac{R_{31}}{R_{32}} \tag{4.7}$$

where R_{ij} are the matrix entries of $\mathbf{R}$.

The rotation matrix and Euler angle conversions are given in the routines `euler2rot` and `rot2euler`. A plotting function, `plotrot`, not copied here, will plot the axes of the rotated frame given $\mathbf{R}$.

```
function [R] = euler2rot(alpha,beta,gamma)
% 3D rotation matrix for ZXZ Euler angles
%
% alpha:    right-handed rotation angle about Z (radians)
% beta:     right-handed rotation angle about X (radians)
% gamma:    right-handed rotation angle about Z (radians)
%
% R:        [3x3] rotation matrix

    Rza = [cos(alpha) -sin(alpha) 0; sin(alpha) cos(alpha) 0; 0 0 1];
    Rxb = [1 0 0; 0 cos(beta) -sin(beta); 0 sin(beta) cos(beta)];
    Rzg = [cos(gamma) -sin(gamma) 0; sin(gamma) cos(gamma) 0; 0 0 1];
    R = Rza*Rxb*Rzg;
end
```

```
function [alpha beta gamma] = rot2euler(R)
% Convert a 3x3 rotation matrix to the Euler angles alpha, beta, gamma
%
% R:                3x3 rotation matrix
%
% alpha,beta,gamma: ZXZ Euler angles (randians)

    alpha = atan2(R(1,3),-R(2,3));
    beta = atan2(sqrt(R(1,3)^2+R(2,3)^2),R(3,3));
    gamma = atan2(R(3,1),R(3,2));
end
```

4.2 Rotation Addition Theorem

The rotation addition theorem for spherical harmonics is given by

$$Y_{lm}(\theta,\phi) = \sum_{p=-l}^{l} D_{lmp}(\alpha,\beta,\gamma) Y_{lp}(\theta',\phi') \tag{4.8}$$

where (α,β,γ) are the three Euler angles describing a ZXZ rotation from unprimed to primed coordinate systems. D_{lmp} is the rotation matrix for spherical harmonics. Because the coordinate r and the gradient are invariant under rotation, the rotation addition theorem for scalar and vector wave functions have identical forms [14, 15, 16, 17],

$$\psi_{lm}(r,\theta,\phi) = \sum_{p=-l}^{l} D_{lmp}(\alpha,\beta,\gamma) \psi_{lp}(r',\theta',\phi') \tag{4.9}$$

$$\mathbf{M}_{lm}(r,\theta,\phi) = \sum_{p=-l}^{l} D_{lmp}(\alpha,\beta,\gamma) \mathbf{M}'_{lp}(r',\theta',\phi') \tag{4.10}$$

This means that the same D_{lmp} can be used to rotate any of following: spherical harmonics, scalar spherical wave functions, vector spherical harmonics, or vector spherical wave functions. D_{lmp} is a square block diagonal matrix with blocks that span m and p for a given l, zero otherwise. Specifically,

$$D_{lmp}(\alpha,\beta,\gamma) = e^{im\alpha} d_{lmp}(\beta) e^{ip\gamma} \tag{4.11}$$

The rotation matrix is inherently separable in (α,β,γ), where α, and γ are diagonal. The matrix $d_{lmp}(\beta)$, called "little-d", is block diagonal and can be computed itself. This is advantageous when precomputing rotation matrices over many combinations of Euler angles. The matrix $d_{lmp}(\beta)$ is given by

$$d_{lmp}(\beta) = i^{m-p} \sqrt{\frac{(l+p)!(l-p)!}{(l+m)!(l-m)!}} \sum_s (-1)^s \binom{l+m}{l+p-s} \binom{l-m}{s} \left(\cos\frac{\beta}{2}\right)^{2l+p-m-2s} \left(\sin\frac{\beta}{2}\right)^{m-p+2s} \tag{4.12}$$

The sum is over all s for positive arguments of the binomial coefficients. There is a version used in quantum mechanics, [18], based on a ZYZ rotation that yields a real-valued $d_{lmp}(\beta)$ matrix. This forms the basis of a recursion algorithm in Section 4.4. The mapping from complex ZXZ $d_{lmp}(\beta)$ to the purely real ZYZ is, [19],

$$d^{ZYZ}_{lmp}(\beta) = (-i)^{p-m} d^{ZXZ}_{lmp}(\beta) \tag{4.13}$$

4.2.1 Field Rotations

Given a field expansion in an unprimed system,

$$\mathbf{E}(\mathbf{r}) = \sum_{lm} a_{lm} \mathbf{M}_{lm}(k,\mathbf{r}) + b_{lm} \mathbf{N}_{lm}(k,\mathbf{r}) \tag{4.14}$$

the expansion coefficients in the primed system are found by substituting the addition theorem

$$\mathbf{E}'(\mathbf{r}') = \sum_{lp} a'_{lp} \mathbf{M}'_{lp}(k,\mathbf{r}') + b'_{lp} \mathbf{N}'_{lp}(k,\mathbf{r}') \tag{4.15}$$

where

$$a'_{lp} = \sum_m a_{lm} D_{lmp}, \qquad b'_{lp} = \sum_m b_{lm} D_{lmp} \tag{4.16}$$

In matrix form, if $\mathbf{a}$ are the outgoing wave coefficients in the i frame, and $\mathbf{a}'$ are those in the rotated i' frame, then

$$\mathbf{a}' = \mathbf{D}_i \mathbf{a} \tag{4.17}$$

where $\mathbf{D}_i$ is the rotation matrix, and (α, β, γ) describe the rotation from i to i'. In other words, the application of $\mathbf{D}_i$ is an inverse rotation on the field (the coefficients $\mathbf{a}'$ describe the same field but viewed from the rotated frame). $\mathbf{D}_i$ is unitary, so the inverse is the conjugate transpose: $\mathbf{D}_i^{-1} = \mathbf{D}_i^*$. This is the forward rotation, where the field itself rotates, to follow the Euler angles, as seen from the originating frame.

4.2.2 Properties

Here we give several properties of the rotation matrix. The transpose satisfies the following relation

$$D_{lmp}(\alpha, \beta, \gamma) = D^*_{lpm}(-\gamma + \pi, \beta, -\alpha + \pi) \tag{4.18}$$

The following relation exists for conjugated matrix elements

$$D_{l,-m,-p} = (-1)^{m+p} D^*_{lmp} \tag{4.19}$$

The following identity exists between the $d_{lmp}(\beta)$ and the Legendre polynomials

$$d_{l00}(\beta) \quad = \quad P_l(\cos\beta) \tag{4.20}$$

The rotation matrix is orthogonal over the three Euler angles as

$$\int_0^{2\pi}\int_0^{\pi}\int_0^{2\pi} D^*_{l'm'p'}(\alpha,\beta,\gamma) D_{lmp}(\alpha,\beta,\gamma) d\alpha \sin\beta d\beta d\gamma = \frac{8\pi^2}{2l+1}\delta_{l'l}\delta_{m'm}\delta_{p'p} \tag{4.21}$$

where

$$\int_0^{2\pi} d\alpha \int_0^{\pi} \sin\beta d\beta \int_0^{2\pi} d\gamma = 8\pi^2 \tag{4.22}$$

The same integral computed over a matrix element is

$$\begin{aligned} I_{lmp} &= \int_0^{2\pi}\int_0^{\pi}\int_0^{2\pi} D_{lmp}(\alpha,\beta,\gamma) d\alpha \sin\beta d\beta d\gamma && (4.23)\\ &= 4\pi^2\delta_{m,0}\delta_{p,0}\int_0^{\pi} d_{lmp}(\beta)\sin\beta d\beta && (4.24)\\ &= 4\pi^2\delta_{m,0}\delta_{p,0}\int_0^{\pi} P_l(\cos\beta)\sin\beta d\beta && (4.25)\\ &= 8\pi^2\delta_{l,0}\delta_{m,0}\delta_{p,0} && (4.26) \end{aligned}$$

For $\alpha = 0$, $\gamma = 0$, the sum over the diagonal elements of the blocks satisfy

$$\sum_{m=-l}^{l} D_{lmm}(\beta) = \sum_{m=-l}^{l} d_{lmm}(\beta) = \frac{\sin\left(\frac{(2l+1)}{2}\beta\right)}{\sin\left(\frac{\beta}{2}\right)} \tag{4.27}$$

which is the normalized Dirichlet kernel.

4.3 Computation of D_{lmp}

Direct computation of the rotation matrix using (4.11) and (4.12) is not advised because of the need to compute the binomial coefficients. Instead, a fast recursion algorithm for computing the full D_{lmp} is given in [20]. This begins with a rotation matrix defined

$$(\hat{x},\hat{y},\hat{z}) = (x,y,z)\begin{bmatrix} R_{xx} & R_{xy} & R_{xz} \\ R_{yx} & R_{yy} & R_{yz} \\ R_{zx} & R_{zy} & R_{zz} \end{bmatrix} = (x,y,z)\mathbf{R} \tag{4.28}$$

which is opposite to the rotation matrix in (4.2). Next, [20] defines the following matrices

$$\mathbf{D} = \mathbf{F} + i\mathbf{G} \tag{4.29}$$

$$\mathbf{F} = \begin{bmatrix} (R_{yy}+R_{xx})/2 & R_{xz}/\sqrt{2} & (R_{yy}-R_{xx})/2 \\ R_{zx}/\sqrt{2} & R_{zz} & -R_{zx}/\sqrt{2} \\ (R_{yy}-R_{xx})/2 & -R_{xz}/\sqrt{2} & (R_{yy}+R_{xx})/2 \end{bmatrix} \tag{4.30}$$

$$\mathbf{G} = \begin{bmatrix} (R_{yx}-R_{xy})/2 & R_{yz}/\sqrt{2} & -(R_{yx}+R_{xy})/2 \\ -R_{zy}/\sqrt{2} & 0 & -R_{zy}/\sqrt{2} \\ (R_{yx}+R_{xy})/2 & R_{yz}/\sqrt{2} & (R_{xy}-R_{yx})/2 \end{bmatrix} \tag{4.31}$$

which serves as the initial condition

$$D_{1mp} = \mathbf{D} \tag{4.32}$$

The recursion equations are given as follows:

1. For $(-l+1) \le p \le (l-1)$:

$$D_{lmp} = a_{lmp}D_{100}D_{l-1,mp} + b_{lmp}D_{110}D_{l-1,m-1,p} + b_{l,-m,p}D_{1,-10}D_{l-1,m+1,p} \tag{4.33}$$

$$a_{lmp} = \sqrt{\frac{(l+m)(l-m)}{(l+p)(l-p)}} \tag{4.34}$$

$$b_{lmp} = \sqrt{\frac{(l+m)(l+m-1)}{2(l+p)(l-p)}} \tag{4.35}$$

$p = \pm l$ is not covered, $a_{lmp} = 0$ for $m = \pm l$, and $b_{lmp} = 0$ for both $m = -l$ and $m = -l+1$.

2. For $-l \le p \le (l-2)$:

$$\begin{aligned} D_{lmp} = {} & c_{lm,-p}D_{10,-1}D_{l-1,m,p+1} + d_{lm,-p}D_{11,-1}D_{l-1,m-1,p+1} \\ & + d_{l,-m,-p}D_{1,-1,-1}D_{l-1,m+1,p+1} \end{aligned} \tag{4.36}$$

$$c_{lmp} = \sqrt{\frac{2(l+m)(l-m)}{(l+p)(l+p-1)}} \tag{4.37}$$

$$d_{lmp} = \sqrt{\frac{(l+m)(l+m-1)}{(l+p)(l+p-1)}} \tag{4.38}$$

$p = \pm l$ and $p = (l-1)$ are not covered, $c_{lmp} = 0$ for $m = \pm l$, and $d_{lmp} = 0$ for both $m = -l$ and $m = -l+1$.

3. For $(-l+1) \le p \le l$:

$$\begin{aligned} D_{lmp} &= c_{lm,p} D_{101} D_{l-1,m,p-1} + d_{lm,p} D_{111} D_{l-1,m-1,p-1} \\ &\quad + d_{l,-m,p} D_{1,-1,1} D_{l-1,m+1,p-1} \end{aligned} \tag{4.39}$$

$p = -l$ and $p = -l+1$ are not covered, and c_{lmp} and d_{lmp} are the same as above.

This algorithm is the basis for two routines that follow. The first routine computes `Dlmp` on a full square matrix, which has mostly zero elements. The second routine is specialized to compute this on a sparse 1D array to same memory.

4.3.1 Full D_{lmp} Matrix

The routine `Dlmp` returns the rotation matrix in full, including zeros, starting with $l = 1$. As mentioned, this describes an inverse rotation. Use the string switch `'mono'` to include the monopole term. The algorithm above uses a rotation matrix that is transposed from our convention, so we transpose it on input to keep the routine consistent. There are helper functions to compute the coefficients: `a_lmp`, `b_lmp`, `c_lmp`, `d_lmp`. Figure 4.2 shows the magnitude of D_{lmp} for different β (α, γ do not affect the magnitude).

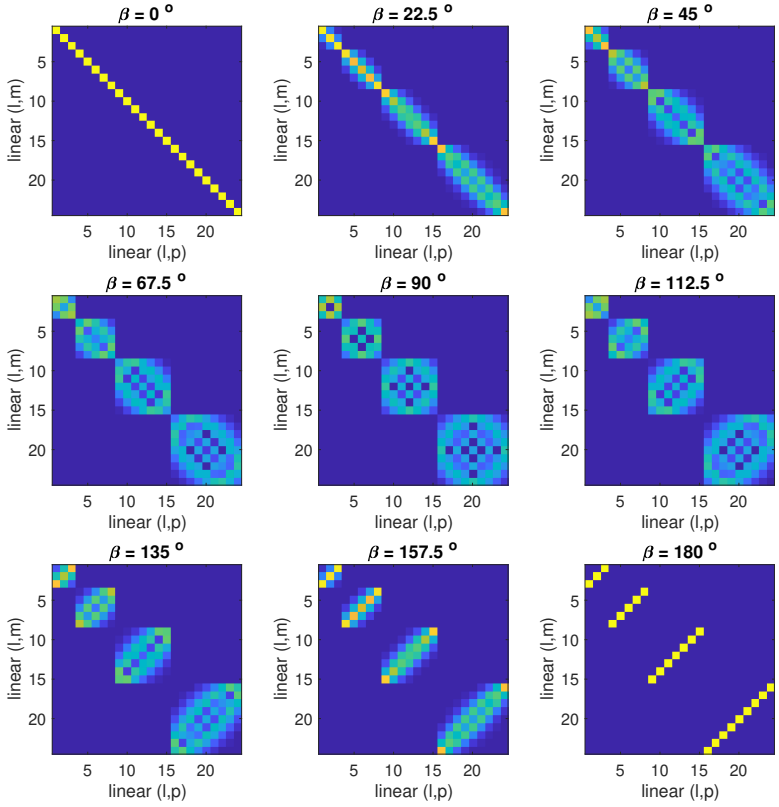


Figure 4.2: D_{lmp} magnitude (color scale [0, 1]) for varying β, $\alpha = \gamma = 0$.

```
function [Dlmp] = Dlmp(L,alpha,beta,gamma,str)
% Rotation addition theorem for spherical harmonics
%
% Y_l^m(theta,phi) = sum_{l,p} D_{l,m,p} Y_l^p(theta',phi')
%
% l = 1:L, for use with vector wave functions (no monopole).
% (theta', phi') are angles seen from the rotated system.
% Rotation defined from unprimed to primed given by ZXZ
% Euler angles alpha, beta, gamma
```

```
%
% L:            Largest harmonic l, with l = 1:L, or l = 0:L
% alpha:        Z Euler angle (rad)
% beta:         X Euler angle (rad)
% gamma:        Z Euler angle (rad)
% str:          [optional] 'mono' for monopole
%
% Dlmp:         NxN Block diagonal rotation matrix
%               N = L^2 + 2*L
%               (l,m) index = l^2 + l + m
%               ...monopole
%               N = L^2 + 2*L + 1
%               (l,m) index = l^2 + l + m + 1
%
% Dependencies:     euler2rot,lm2ind,a_lmp,b_lmp,c_lmp,d_lmp

if L == 0
    Dlmp = 1;
    return
elseif L < 1
    error('bad L')
end
[Rot] = euler2rot(alpha,beta,gamma);
Rot = Rot.';
rxx = Rot(1,1);
rxy = Rot(1,2);
rxz = Rot(1,3);
ryx = Rot(2,1);
ryy = Rot(2,2);
ryz = Rot(2,3);
rzx = Rot(3,1);
rzy = Rot(3,2);
rzz = Rot(3,3);
F = [(ryy+rxx)/2 rxz/sqrt(2) (ryy-rxx)/2;
    rzx/sqrt(2) rzz -rzx/sqrt(2);
    (ryy-rxx)/2 -rxz/sqrt(2) (ryy+rxx)/2];
G = [(ryx-rxy)/2 ryz/sqrt(2) -(ryx+rxy)/2;
    -rzy/sqrt(2) 0 -rzy/sqrt(2);
    (ryx+rxy)/2 ryz/sqrt(2) (rxy-ryx)/2];
D1 = F+1i*G;
tot = L^2 + 2*L;
Dlmp = zeros(tot);
Dlmp(1:3,1:3) = D1;
for l=2:L,
    for m = (-l+1):(l-1),
    for p = (-l+1):(l-1),
        a1 = a_lmp(l,m,p);
        b1 = b_lmp(l,m,p);
        b2 = b_lmp(l,-m,p);
        if a1~=0
            tmp1 = a1*D1(2,2)*Dlmp(lm2ind(l-1,m),lm2ind(l-1,p));
        else tmp1 = 0; end
        if b1~=0
            tmp2 = b1*D1(3,2)*Dlmp(lm2ind(l-1,m-1),lm2ind(l-1,p));
        else tmp2 = 0; end
        if b2~=0
            tmp3 = b2*D1(1,2)*Dlmp(lm2ind(l-1,m+1),lm2ind(l-1,p));
        else tmp3 = 0; end
        inm = lm2ind(l,m);
        inp = lm2ind(l,p);
        Dlmp(inm,inp) = tmp1+tmp2+tmp3;
    end
    end
    m = -l;
    for p = (-l+1):(l-1),
        b2 = b_lmp(l,-m,p);
        Dlmp(lm2ind(l,m),lm2ind(l,p)) = ...
            b2*D1(1,2)*Dlmp(lm2ind(l-1,m+1),lm2ind(l-1,p));
```

```
    end
    m = l;
    for p = (-l+1):(l-1),
        b1 = b_lmp(l,m,p);
        Dlmp(lm2ind(l,m),lm2ind(l,p)) = ...
            b1*D1(3,2)*Dlmp(lm2ind(l-1,m-1),lm2ind(l-1,p));
    end
    p = -l;
    for m = (-l):(l),
        c1 = c_lmp(l,m,-p);
        d1 = d_lmp(l,m,-p);
        d2 = d_lmp(l,-m,-p);
        if c1~=0
            tmp1 = c1*D1(2,1)*Dlmp(lm2ind(l-1,m),lm2ind(l-1,p+1));
        else tmp1 = 0; end
        if d1~=0
            tmp2 = d1*D1(3,1)*Dlmp(lm2ind(l-1,m-1),lm2ind(l-1,p+1));
        else tmp2 = 0; end
        if d2~=0
            tmp3 = d2*D1(1,1)*Dlmp(lm2ind(l-1,m+1),lm2ind(l-1,p+1));
        else tmp3 = 0; end
        Dlmp(lm2ind(l,m),lm2ind(l,p)) = tmp1+tmp2+tmp3;
    end
    p = l;
    for m = (-l):(l),
        c1 = c_lmp(l,m,p);
        d1 = d_lmp(l,m,p);
        d2 = d_lmp(l,-m,p);
        if c1~=0
            tmp1 = c1*D1(2,3)*Dlmp(lm2ind(l-1,m),lm2ind(l-1,p-1));
        else tmp1 = 0; end
        if d1~=0
            tmp2 = d1*D1(3,3)*Dlmp(lm2ind(l-1,m-1),lm2ind(l-1,p-1));
        else tmp2 = 0; end
        if d2~=0
            tmp3 = d2*D1(1,3)*Dlmp(lm2ind(l-1,m+1),lm2ind(l-1,p-1));
        else tmp3 = 0; end
        Dlmp(lm2ind(l,m),lm2ind(l,p)) = tmp1+tmp2+tmp3;
    end
end
if nargin == 5 && ~isempty(str)
    if ~strcmp(str,'mono')
        error('bad string')
    end
    tot = L^2+2*L+1;
    dlmp2 = zeros(tot);
    dlmp2(2:end,2:end) = Dlmp;
    Dlmp = dlmp2;
    Dlmp(1) = 1;
end
end

% Helper functions
function a = a_lmp(l,m,p)
    a = sqrt((l+m)*(l-m)/((l+p)*(l-p)));
end
function b = b_lmp(l,m,p)
    b = sqrt((l+m)*(l+m-1)/(2*(l+p)*(l-p)));
end
function c = c_lmp(l,m,p)
    c = sqrt(2*(l+m)*(l-m)/((l+p)*(l+p-1)));
end
function d = d_lmp(l,m,p)
    d = sqrt((l+m)*(l+m-1)/((l+p)*(l+p-1)));
end
```

4.3.2 Sparse D_{lmp} Matrix

The rotation matrix that is the output of `Dlmp` should not be used directly for matrix-vector multiplication because it contains unnecessary zeros. The quick fix is to immediately transform it to a sparse matrix:

```
D = sparse(Dlmp(L,alpha,beta,gamma));
```

If several different rotation matrices are needed, they will likely have different numbers of non-zero elements, which can lead to problems with preallocation. Furthermore, the routine `Dlmp` works on a large, mostly zero, 2D matrix. Therefore, we have a modified version of the routine `Dlmp`, which recurses on a sparse matrix stored as a 1D array. The modifications follow.

Because the rotation matrix is square block diagonal, the maximum number of non-zero elements (excluding the monopole) is found by counting the elements in blocks that are sized $2l + 1$ on a side up to a maximum degree harmonic L:

$$N_L = \sum_{l=1}^{L} (2l+1)^2 \tag{4.40}$$

$$= 4\sum_{l=1}^{L} l^2 + 4\sum_{l=1}^{L} l + \sum_{l=1}^{L} 1 \tag{4.41}$$

$$= \frac{4}{3}L^3 + 4L^2 + \frac{11}{3}L \tag{4.42}$$

Add 1 for the monopole. Next, we map the 2D row/column linear indices to a 1D index of the sparse matrix. Let the row linear index for the (l, m) harmonic in the full matrix be $l^2 + l + m$ and let the column linear index of the (l, p) harmonic in the full matrix be $l^2 + l + p$. Counting column-major (top-down, left-right), the linear index for the (l, m, p) element in the sparse array is

$$I(l, m, p) = N_{l-1} + (2l + 1)(l + p) + (l + m + 1) \tag{4.43}$$

The term N_{l-1} counts the number of elements for all full blocks less than l, the second term counts the number of elements in full columns of the current block less than p, the last term counts the rows up to m in the current column. Again, add 1 for the monopole.

The routine `DlmpSparse` returns the three column arrays containing the row index, column index, and matrix entries of D_{lmp}. It is the same as `Dlmp` except computed on a 1D array. Use the string switch `'mono'` for the monopole. The two helper functions `NDlmpSparse` and `indDlmpSparse` return the number of elements and index for the sparse rotation matrix. A further refinement could use conjugate symmetry to only store half of the matrix plus the main diagonal, but that then requires specialized routines for matrix-vector multiplication. Finally, the helper functions as written create a bottleneck, so a faster version, `DlmpSparseFast`, is included but not copied here. This uses inline indexing and is about five times faster for moderate values of L.

```
function [row col Dlmp] = DlmpSparse(L,alpha,beta,gamma,str)
% Rotation addition theorem for spherical harmonics, sparse matrix
%
% Y_l^m(theta,phi) = sum_{l,p} D_{l,m,p} Y_l^p(theta',phi')
%
% l = 1:L, for use with vector wave functions (no monopole).
% (theta', phi') are angles seen from the rotated system.
% Rotation defined from unprimed to primed given by ZXZ
% Euler angles alpha, beta, gamma
%
% L:            Largest harmonic l, with l = 1:L, or l = 0:L
% alpha:        Z Euler angle (rad)
% beta:         X Euler angle (rad)
% gamma:        Z Euler angle (rad)
% str:          'mono' for monopole
%
% row,col:      row and column indices of matrix entries
% Dlmp:         Sparse rotation matrix entries
```

```
%
% Dependencies:     euler2rot,NDlmpSparse,indDlmpSparse,lm2ind,a_lmp,b_lmp,c_lmp,d_lmp

if L == 0
    Dlmp = 1;
    return
elseif L < 1
    disp('bad L')
    return
end
[Rot] = euler2rot(alpha,beta,gamma);
Rot = Rot.';
rxx = Rot(1,1);
rxy = Rot(1,2);
rxz = Rot(1,3);
ryx = Rot(2,1);
ryy = Rot(2,2);
ryz = Rot(2,3);
rzx = Rot(3,1);
rzy = Rot(3,2);
rzz = Rot(3,3);
F = [(ryy+rxx)/2 rxz/sqrt(2) (ryy-rxx)/2;
    rzx/sqrt(2) rzz -rzx/sqrt(2);
    (ryy-rxx)/2 -rxz/sqrt(2) (ryy+rxx)/2];
G = [(ryx-rxy)/2 ryz/sqrt(2) -(ryx+rxy)/2;
    -rzy/sqrt(2) 0 -rzy/sqrt(2);
    (ryx+rxy)/2 ryz/sqrt(2) (rxy-ryx)/2];
D1 = F+1i*G;

NL = NDlmpSparse(L);
Dlmp = zeros(NL,1);
row = zeros(NL,1);
col = zeros(NL,1);

l=1;
for m=-l:l,
for p=-l:l,
    inm = lm2ind(l,m);
    inp = lm2ind(l,p);
    inS = indDlmpSparse(l,m,p);
    row(inS) = inm;
    col(inS) = inp;
    Dlmp(inS) = D1(inm,inp);
end
end

for l=2:L,
    for m = (-l+1):(l-1),
    for p = (-l+1):(l-1),
        a1 = a_lmp(l,m,p);
        b1 = b_lmp(l,m,p);
        b2 = b_lmp(l,-m,p);
        if a1~=0
            tmp1 = a1*D1(2,2)*Dlmp(indDlmpSparse(l-1,m,p));
        else tmp1 = 0; end
        if b1~=0
            tmp2 = b1*D1(3,2)*Dlmp(indDlmpSparse(l-1,m-1,p));
        else tmp2 = 0; end
        if b2~=0
            tmp3 = b2*D1(1,2)*Dlmp(indDlmpSparse(l-1,m+1,p));
        else tmp3 = 0; end
        inS = indDlmpSparse(l,m,p);
        row(inS) = lm2ind(l,m);
        col(inS) = lm2ind(l,p);
        Dlmp(inS) = tmp1+tmp2+tmp3;
    end
    end
    m = -l;
```

```
    for p = (-l+1):(l-1),
        b2 = b_lmp(l,-m,p);
        inS = indDlmpSparse(l,m,p);
        row(inS) = lm2ind(l,m);
        col(inS) = lm2ind(l,p);
        Dlmp(inS) = b2*D1(1,2)*Dlmp(indDlmpSparse(l-1,m+1,p));
    end
    m = l;
    for p = (-l+1):(l-1),
        b1 = b_lmp(l,m,p);
        inS = indDlmpSparse(l,m,p);
        row(inS) = lm2ind(l,m);
        col(inS) = lm2ind(l,p);
        Dlmp(inS) = b1*D1(3,2)*Dlmp(indDlmpSparse(l-1,m-1,p));
    end
    p = -l;
    for m = (-l):(l),
        c1 = c_lmp(l,m,-p);
        d1 = d_lmp(l,m,-p);
        d2 = d_lmp(l,-m,-p);
        if c1~=0
            tmp1 = c1*D1(2,1)*Dlmp(indDlmpSparse(l-1,m,p+1));
        else tmp1 = 0; end
        if d1~=0
            tmp2 = d1*D1(3,1)*Dlmp(indDlmpSparse(l-1,m-1,p+1));
        else tmp2 = 0; end
        if d2~=0
            tmp3 = d2*D1(1,1)*Dlmp(indDlmpSparse(l-1,m+1,p+1));
        else tmp3 = 0; end
        inS = indDlmpSparse(l,m,p);
        row(inS) = lm2ind(l,m);
        col(inS) = lm2ind(l,p);
        Dlmp(inS) = tmp1+tmp2+tmp3;
    end
    p = l;
    for m = (-l):(l),
        c1 = c_lmp(l,m,p);
        d1 = d_lmp(l,m,p);
        d2 = d_lmp(l,-m,p);
        if c1~=0
             tmp1 = c1*D1(2,3)*Dlmp(indDlmpSparse(l-1,m,p-1));
        else tmp1 = 0; end
        if d1~=0
             tmp2 = d1*D1(3,3)*Dlmp(indDlmpSparse(l-1,m-1,p-1));
        else tmp2 = 0; end
        if d2~=0
            tmp3 = d2*D1(1,3)*Dlmp(indDlmpSparse(l-1,m+1,p-1));
        else tmp3 = 0; end

        inS = indDlmpSparse(l,m,p);
        row(inS) = lm2ind(l,m);
        col(inS) = lm2ind(l,p);
        Dlmp(inS) = tmp1+tmp2+tmp3;
    end
end
if nargin == 5 && strcmp(str,'mono')
    NL = NDlmpSparse(L,'mono');
    Dlmp2 = zeros(NL,1);
    Dlmp2(2:end) = Dlmp;
    Dlmp = Dlmp2;
    Dlmp(1) = 1;
    row2 = zeros(NL,1);
    row2(2:end) = row + 1;
    row = row2;
    row(1) = 1;
    col2 = zeros(NL,1);
    col2(2:end) = col + 1;
    col = col2;
```

```
    col(1) = 1;
end
end

% Helper functions
function a = a_lmp(l,m,p)
    a = sqrt((l+m)*(l-m)/((l+p)*(l-p)));
end
function b = b_lmp(l,m,p)
    b = sqrt((l+m)*(l+m-1)/(2*(l+p)*(l-p)));
end
function c = c_lmp(l,m,p)
    c = sqrt(2*(l+m)*(l-m)/((l+p)*(l+p-1)));
end
function d = d_lmp(l,m,p)
    d = sqrt((l+m)*(l+m-1)/((l+p)*(l+p-1)));
end


function [Nl] = NDlmpSparse(L,str)
% Number of non-zero matrix entries for Dlmp rotation matrix
%
% L:    Maximum harmonic L
% str:  'mono' for monopole
%
% Nl:   Number of non-zero matrix entries

if nargin == 2 && strcmp(str,'mono')
    if L < 0, error('bad index'), end
    Nl = round(4/3*L^3 + 4*L^2 + 11/3*L) + 1;
else
    if L < 1, error('bad index'), end
    Nl = round(4/3*L^3 + 4*L^2 + 11/3*L);
end


function [ind] = indDlmpSparse(l,m,p,str)
% Returns linear index for sparse rotation matrix Dlmp
%
% l,m,p:    angular harmonic indices
% str:      'mono' for monopole
%
% ind:      sparse matrix linear index

if nargin == 4 && strcmp(str,'mono')
    if l < 0 || abs(m) > l
        error('bad index')
    end
    if l == 0
        ind = 1;
        return
    else
        c1 = (2*l+1)*(l+p);
        c2 = l+m+1;
        ind = NDlmpSparse(l-1,str) + round(c1 + c2);
    end
else
    if l < 1 || abs(m) > l
        error('bad index')
    end
    c1 = (2*l+1)*(l+p);
    c2 = l+m+1;
    if l == 1
        ind = round(c1 + c2);
    else
        ind = NDlmpSparse(l-1) + round(c1 + c2);
    end
end
```

4.4 Computation of $d_{lmp}(\beta)$

This section contains routines for the computation of "little-d", $d_{lmp}(\beta)$. This rotation matrix should be used with (4.11) to precompute and loop over combinations of Euler angles. Four routines are provided. The first is a direct computation for cross-checking, but should not be used in practice because computing the factorials in the binomial coefficients is not accurate. The second is based on a fast recursion of the purely real ZYZ $d_{lmp}(\beta)$, computed on the full matrix, then converted to complex ZXZ $d_{lmp}(\beta)$. The third routine is the same as the second but faster. The last routine is the same as the second but computed on the sparse matrix using the sparse indexing tools developed before.

4.4.1 Direct Computation

The routine `dlmpBetaDirect` computes the ZXZ complex $d_{lmp}(\beta)$ directly from (4.12). This is good to maybe $L = 20$ due to direct computation of the factorials. This is used to validate the recursive algorithms.

```
function dlmp = dlmpBetaDirect(L,beta,str)
% Direct computation of 'little-d' rotation matrix
%
% L:        maximum harmonic L
% beta:     Euler angle about X
% str:      'mono' to include monopole
%
% dlmp:     'little-d' rotation matrix
%
% Dependencies: lm2ind

N = L^2 + 2*L;
dlmp = zeros(N,N);
for l=1:L,
for m=-l:l,
for p=-l:l,
    row = lm2ind(l,m);
    col = lm2ind(l,p);
    f1 = factorial(l+p);
    f2 = factorial(l-p);
    f3 = factorial(l+m);
    f4 = factorial(l-m);
    const = sqrt(f1*f2/(f3*f4));
    tmp = 0;
    arg1 = l+m;
    arg2 = l-m;
    for u=0:(l+p),
       k1 = l+p-u;
       k2 = u;
       if k1 >= 0 && k1 <= arg1 && k2 >= 0 && k2 <= arg2
           t1=nchoosek(l+m,l+p-u);
           t2=nchoosek(l-m,u);
           signm = ((-1)^u)*((1i)^(m-p));
           c1 = (cos(beta/2)).^(2*l+p-m-2*u);
           s1 = (sin(beta/2)).^(m-p+2*u);
           tmp = tmp + signm*t1*t2*c1*s1;
       end
    end
    dlmp(row,col) = const*tmp;
end
end
end

if nargin == 3 && strcmp(str,'mono')
    tot = L^2+2*L+1;
    dlmp2 = zeros(tot);
    dlmp2(2:end,2:end) = dlmp;
    dlmp = dlmp2;
    dlmp(1) = 1;
end
```

4.4.2 Recursion Algorithm

A fast computation of $d_{lmp}(\beta)$ is based on the recursive algorithm in [21], which gives an algorithm for the purely real ZYZ matrix. The recursion equations for real $d_{lmp}(\beta)$ are

$$d_{lmp} = \cos^2(x)A_{lmp}d_{l-1,m-1,p-1} - 2\sin(x)\cos(x)B_{lmp}d_{l-1,m-1,p} + \sin^2(x)C_{lmp}d_{l-1,m-1,p+1} \quad (4.44)$$

$$d_{lmp} = \sin^2(x)D_{lmp}d_{l-1,m+1,p-1} + 2\sin(x)\cos(x)E_{lmp}d_{l-1,m+1,p} + \cos^2(x)F_{lmp}d_{l-1,m+1,p+1} \quad (4.45)$$

$$d_{lmp} = \sin(x)\cos(x)G_{lmp}d_{l-1,m,p-1} + (\cos^2(x) - \sin^2(x))H_{lmp}d_{l-1,m,p} - \sin(x)\cos(x)I_{lmp}d_{l-1,m,p+1} \quad (4.46)$$

where $x = \beta/2$. The coefficients can be written as an outer product

$$\begin{bmatrix} A^2_{lmp} & B^2_{lmp} & C^2_{lmp} \\ D^2_{lmp} & E^2_{lmp} & F^2_{lmp} \\ G^2_{lmp} & H^2_{lmp} & I^2_{lmp} \end{bmatrix} = \begin{bmatrix} \frac{1}{(l+m)(l+m-1)} \\ \frac{1}{(l-m)(l-m-1)} \\ \frac{1}{(l-m)(l+m)} \end{bmatrix} \begin{bmatrix} (l+p)(l+p-1) & (l+p)(l-p) & (l-p)(l-p-1) \end{bmatrix} \quad (4.47)$$

The recursion algorithm first computes (4.44) for $m = l$ and all $\pm p$, then (4.45) for $m = -l$ all p, and finally (4.46) for $m = (-l+1), ..., (l-1)$ all p. When a coefficient is zero, the corresponding term at $l-1$ does not exist and can be ignored. With the conversion to complex using (4.13), this recursion matches the direct compuation and matches the full `Dlmp` for $(\alpha, \beta, \gamma) = (0, \beta, 0)$. Note, in [21], the numerator of I_{lmp} has $(l-p+1)$, which is incorrect. It should be $(l-p-1)$, so that it preserves the symmetry of the coefficients.

The routine `dlmpBeta` is a straight implementation of (4.44), (4.45), (4.46) on the full matrix (with zeros) with conversion to complex. Use `'mono'` to include the monopole. A second version of this routine, `dlmpBetaFast`, which is not copied here but included in the library, is a faster implementation that uses inline indexing, inline coefficient computation, and vectorized real to complex conversion. Still on the full matrix, it runs about six times faster.

```
function dlmp = dlmpBeta(L,beta,str)
% Recursive algorithm for ZXZ 'little-d' complex rotation matrix
%
% L:        maximum harmonic L
% beta:     Euler angle about X
% str:      'mono' to include monopole
%
% dlmp:     'little-d' rotation matrix
%
% Dependencies: lm2ind

% includes monopole for initial condition
N = L^2 + 2*L + 1;
dlmp = zeros(N,N);
dlmp(1,1) = 1;
x = beta/2;

for l=1:L,
    m=l;
    for p=-l:l
        A = Almp(l,m,p);
        B = Blmp(l,m,p);
        C = Clmp(l,m,p);
        if A~=0
            tmp1 = cos(x)^2*A*...
                dlmp(lm2ind(l-1,m-1,'mono'),lm2ind(l-1,p-1,'mono'));
        else tmp1 = 0; end
        if B~=0
            tmp2 = -2*sin(x)*cos(x)*B*...
                dlmp(lm2ind(l-1,m-1,'mono'),lm2ind(l-1,p,'mono'));
        else tmp2 = 0; end
        if C~=0
```

```
            tmp3 = sin(x)^2*C*...
                dlmp(lm2ind(l-1,m-1,'mono'),lm2ind(l-1,p+1,'mono'));
        else tmp3 = 0; end
        inm = lm2ind(l,m,'mono');
        inp = lm2ind(l,p,'mono');
        dlmp(inm,inp) = (tmp1+tmp2+tmp3);
    end

    m=-l;
    for p=-l:l
        D = Dlmp(l,m,p);
        E = Elmp(l,m,p);
        F = Flmp(l,m,p);
        if D~=0
            tmp1 = sin(x)^2*D*...
                dlmp(lm2ind(l-1,m+1,'mono'),lm2ind(l-1,p-1,'mono'));
        else tmp1 = 0; end
        if E~=0
            tmp2 = 2*sin(x)*cos(x)*E*...
                dlmp(lm2ind(l-1,m+1,'mono'),lm2ind(l-1,p,'mono'));
        else tmp2 = 0; end
        if F~=0
            tmp3 = cos(x)^2*F*...
                dlmp(lm2ind(l-1,m+1,'mono'),lm2ind(l-1,p+1,'mono'));
        else tmp3 = 0; end
        inm = lm2ind(l,m,'mono');
        inp = lm2ind(l,p,'mono');
        dlmp(inm,inp) = (tmp1+tmp2+tmp3);
    end

    for m=(-l+1):(l-1),
    for p=-l:l
        G = Glmp(l,m,p);
        H = Hlmp(l,m,p);
        I = Ilmp(l,m,p);
        if G~=0
            tmp1 = sin(x)*cos(x)*G*...
                dlmp(lm2ind(l-1,m,'mono'),lm2ind(l-1,p-1,'mono'));
        else tmp1 = 0; end
        if H~=0
            tmp2 = (cos(x)^2 - sin(x)^2)*H*...
                dlmp(lm2ind(l-1,m,'mono'),lm2ind(l-1,p,'mono'));
        else tmp2 = 0; end
        if I~=0
            tmp3 = -sin(x)*cos(x)*I*...
                dlmp(lm2ind(l-1,m,'mono'),lm2ind(l-1,p+1,'mono'));
        else tmp3 = 0; end
        inm = lm2ind(l,m,'mono');
        inp = lm2ind(l,p,'mono');
        dlmp(inm,inp) = (tmp1+tmp2+tmp3);
    end
    end


end

% convert to complex
for l=1:L,
for m=-l:l,
for p=-l:l,
    inm = lm2ind(l,m,'mono');
    inp = lm2ind(l,p,'mono');
    dlmp(inm,inp) = ((1i)^(m-p)*(-1)^(m-p))*dlmp(inm,inp);
end
end
end

% no monopole
```

```
if nargin == 2
   dlmp = dlmp(2:end,2:end);
end
end

function a = Almp(l,m,p)
    a = sqrt((l+p)*(l+p-1)/((l+m)*(l+m-1)));
end
function a = Blmp(l,m,p)
    a = sqrt((l+p)*(l-p)/((l+m)*(l+m-1)));
end
function a = Clmp(l,m,p)
    a = sqrt((l-p)*(l-p-1)/((l+m)*(l+m-1)));
end
function a = Dlmp(l,m,p)
    a = sqrt((l+p)*(l+p-1)/((l-m)*(l-m-1)));
end
function a = Elmp(l,m,p)
    a = sqrt((l+p)*(l-p)/((l-m)*(l-m-1)));
end
function a = Flmp(l,m,p)
    a = sqrt((l-p)*(l-p-1)/((l-m)*(l-m-1)));
end
function a = Glmp(l,m,p)
    a = sqrt((l+p)*(l+p-1)/((l-m)*(l+m)));
end
function a = Hlmp(l,m,p)
    a = sqrt((l+p)*(l-p)/((l-m)*(l+m)));
end
function a = Ilmp(l,m,p)
    a = sqrt((l-p)*(l-p-1)/((l-m)*(l+m)));
end
```

4.4.3 Sparse Recursive

Like the full D_{lmp}, $d_{lmp}(\beta)$ can be computed as a sparse matrix on a 1D array with proper bookkeeping to avoid having to create and index a large matrix of zeros. We solved the problem of indexing the sparse rotation matrix before. `dlmpBetaSparse` is a version of `dlmpBeta` that returns the row, column and matrix entries for sparse complex ZXZ $d_{lmp}(\beta)$.

```
function [row col dlmp] = dlmpBetaSparse(L,beta,str)
% Recursive algorithm for ZXZ 'little-d' complex rotation matrix
% computed on sparse matrix
%
% L:        maximum harmonic L
% beta:     Euler angle about X
% str:      'mono' to include monopole
%
% row,col:      row and column indices of matrix entries
% dlmp:         Sparse 'little-d' rotation matrix
%
% Dependencies: lm2ind,NDlmpSparse,indDlmpSparse,

% includes monopole for initial condition
NL = NDlmpSparse(L,'mono');
dlmp = zeros(NL,1);
row = zeros(NL,1);
col = zeros(NL,1);
dlmp(1,1) = 1;
x = beta/2;
for l=1:L,
    m=l;
    for p=-l:l
        A = Almp(l,m,p);
```

```
    B = Blmp(l,m,p);
    C = Clmp(l,m,p);
    if A~=0
        tmp1 = cos(x)^2*A*...
            dlmp(indDlmpSparse(l-1,m-1,p-1,'mono'));
    else tmp1 = 0; end
    if B~=0
        tmp2 = -2*sin(x)*cos(x)*B*...
            dlmp(indDlmpSparse(l-1,m-1,p,'mono'));
    else tmp2 = 0; end
    if C~=0
        tmp3 = sin(x)^2*C*...
            dlmp(indDlmpSparse(l-1,m-1,p+1,'mono'));
    else tmp3 = 0; end
    inS = indDlmpSparse(l,m,p,'mono');
    row(inS) = lm2ind(l,m,'mono');
    col(inS) = lm2ind(l,p,'mono');
    dlmp(inS) = (tmp1+tmp2+tmp3);
end

m=-l;
for p=-l:l
    D = Dlmp(l,m,p);
    E = Elmp(l,m,p);
    F = Flmp(l,m,p);
    if D~=0
        tmp1 = sin(x)^2*D*...
            dlmp(indDlmpSparse(l-1,m+1,p-1,'mono'));
    else tmp1 = 0; end
    if E~=0
        tmp2 = 2*sin(x)*cos(x)*E*...
            dlmp(indDlmpSparse(l-1,m+1,p,'mono'));
    else tmp2 = 0; end
    if F~=0
        tmp3 = cos(x)^2*F*...
            dlmp(indDlmpSparse(l-1,m+1,p+1,'mono'));
    else tmp3 = 0; end
    inS = indDlmpSparse(l,m,p,'mono');
    row(inS) = lm2ind(l,m,'mono');
    col(inS) = lm2ind(l,p,'mono');
    dlmp(inS) = (tmp1+tmp2+tmp3);
end

for m=(-l+1):(l-1),
for p=-l:l
    G = Glmp(l,m,p);
    H = Hlmp(l,m,p);
    I = Ilmp(l,m,p);
    if G~=0
        tmp1 = sin(x)*cos(x)*G*...
            dlmp(indDlmpSparse(l-1,m,p-1,'mono'));
    else tmp1 = 0; end
    if H~=0
        tmp2 = (cos(x)^2 - sin(x)^2)*H*...
            dlmp(indDlmpSparse(l-1,m,p,'mono'));
    else tmp2 = 0; end
    if I~=0
        tmp3 = -sin(x)*cos(x)*I*...
            dlmp(indDlmpSparse(l-1,m,p+1,'mono'));
    else tmp3 = 0; end
    inS = indDlmpSparse(l,m,p,'mono');
    row(inS) = lm2ind(l,m,'mono');
    col(inS) = lm2ind(l,p,'mono');
    dlmp(inS) = (tmp1+tmp2+tmp3);
end
end
```

```
end

% convert to complex
for l=1:L,
for m=-l:l,
for p=-l:l,
    inS = indDlmpSparse(l,m,p,'mono');
    dlmp(inS) = ((1i)^(m-p)*(-1)^(m-p))*dlmp(inS);
end
end
end

% no monopole
if nargin == 2
   row = row(2:end)-1;
   col = col(2:end)-1;
   dlmp = dlmp(2:end);
end
end

function a = Almp(l,m,p)
    a = sqrt((l+p)*(l+p-1)/((l+m)*(l+m-1)));
end
function a = Blmp(l,m,p)
    a = sqrt((l+p)*(l-p)/((l+m)*(l+m-1)));
end
function a = Clmp(l,m,p)
    a = sqrt((l-p)*(l-p-1)/((l+m)*(l+m-1)));
end
function a = Dlmp(l,m,p)
    a = sqrt((l+p)*(l+p-1)/((l-m)*(l-m-1)));
end
function a = Elmp(l,m,p)
    a = sqrt((l+p)*(l-p)/((l-m)*(l-m-1)));
end
function a = Flmp(l,m,p)
    a = sqrt((l-p)*(l-p-1)/((l-m)*(l-m-1)));
end
function a = Glmp(l,m,p)
    a = sqrt((l+p)*(l+p-1)/((l-m)*(l+m)));
end
function a = Hlmp(l,m,p)
    a = sqrt((l+p)*(l-p)/((l-m)*(l+m)));
end
function a = Ilmp(l,m,p)
    a = sqrt((l-p)*(l-p-1)/((l-m)*(l+m)));
end
```

Chapter 5

Translation

This chapter gives routines for computing translation matrices that operate on the spherical wave function expansions given in Chapter 3. Translation matrices, which are derived from addition theorems, allow us to represent a scalar or vector field in two different but parallel reference frames. The frame of reference is being translated, not the field. The addition theorems and matrix diagonalization are first outlined. Next, algorithms and routines for the scalar axial translation matrices (i.e., translation along the z-axis) are given which are computed on full or sparse matrices. Last, algorithms and routines for full and sparse vector axial translation matrices are derived from the scalar versions.

5.1 Translation Addition Theorem

The translation addition theorems for spherical wave functions allow the same field to be expanded in two different, but parallel, frames. These are given for scalar waves as [2],

$$\psi_{l'm'}(k,\mathbf{r}_i) = \sum_{l=0}^{\infty}\sum_{m=-l}^{l} \alpha^{ji}_{lm,l'm'} Rg\psi_{lm}(k,\mathbf{r}_j), \qquad r_j < r_{ji} \tag{5.1}$$

$$\psi_{l'm'}(k,\mathbf{r}_i) = \sum_{l=0}^{\infty}\sum_{m=-l}^{l} \beta^{ji}_{lm,l'm'} \psi_{lm}(k,\mathbf{r}_j), \qquad r_j > r_{ji} \tag{5.2}$$

$$Rg\psi_{l'm'}(k,\mathbf{r}_i) = \sum_{l=0}^{\infty}\sum_{m=-l}^{l} \beta^{ji}_{lm,l'm'} Rg\psi_{lm}(k,\mathbf{r}_j), \qquad \forall r_j \tag{5.3}$$

where $\beta^{ji}_{lm,l'm'} = Rg\alpha^{ji}_{lm,l'm'}$ are translation matrices and Rg is the regular part. The position vectors, $\mathbf{r}_i$ and $\mathbf{r}_j$, are measured from the origins of each frame. The first relation expands radiating waves from frame i as regular or incoming waves in frame j. The second expands radiating waves in frame i as radiating waves in frame j. The last expresses regular waves in frame i and regular waves in frame j. Each addition theorem has a different region of validity: (5.1) is valid within a sphere of radius $r_{ji} = |\mathbf{r}_{ji}| = |\mathbf{r}_j - \mathbf{r}_i|$ centered on frame j; (5.2) is valid outside a radius r_{ji} centered on frame j; (5.3) is valid everywhere.

Translation addition theorems for vector spherical wave functions have similar structures, [2]. The first is

$$\mathbf{M}_{l'm'}(k,\mathbf{r}_i) = \sum_{l=1}^{\infty}\sum_{m=-l}^{l} A^{ji}_{lm,l'm'} Rg\mathbf{M}_{lm}(k,\mathbf{r}_j), + B^{ji}_{lm,l'm'} Rg\mathbf{N}_{lm}(\mathbf{r}_j) \tag{5.4}$$

$$\mathbf{N}_{l'm'}(k,\mathbf{r}_i) = \sum_{l=1}^{\infty}\sum_{m=-l}^{l} B^{ji}_{lm,l'm'} Rg\mathbf{M}_{lm}(k,\mathbf{r}_j), + A^{ji}_{lm,l'm'} Rg\mathbf{N}_{lm}(\mathbf{r}_j) \tag{5.5}$$

where $A^{ji}_{lm,l'm'}$ and $B^{ji}_{lm,l'm'}$ are vector translation matrices and is valid for $r_j < r_{ji}$. The remaining two relations are analogous to the scalar case and can be obtained with appropriate Hankel or Bessel functions in $A^{ji}_{lm,l'm'}$ and $B^{ji}_{lm,l'm'}$. These show that vector modes mix when translated. The spherical vector components

are also transformed and are expressed relative to the origin of the new frame. This means that the vector components in each frame need to be converted to Cartesian components before comparing.

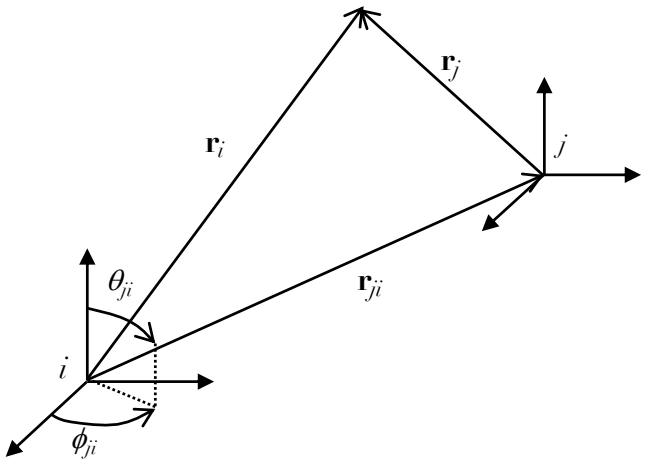


Figure 5.1: Coordinate frames i and j.

Translation matrices allow us to manipulate field using only the expansion coefficients. For example, a radiating scalar field is expanded in frame i as

$$\phi(\mathbf{r}_i) = \sum_{l'=0}^{\infty} \sum_{m'=-l'}^{l'} a_{l'm'} \psi_{l'm'}(\mathbf{r}_i) \tag{5.6}$$

The same field expressed as incoming waves in frame j comes from substituting (5.1) and regrouping terms as

$$\phi(\mathbf{r}_j) = \sum_{l=0}^{\infty} \sum_{m=-l}^{l} b_{lm} Rg\psi_{lm}(k, \mathbf{r}_j) \tag{5.7}$$

$$b_{lm} = \sum_{l'=0}^{\infty} \sum_{m'=-l'}^{l'} \alpha^{ji}_{lm,l'm'} a_{l'm'} \tag{5.8}$$

Likewise, for the other relations. In matrix notation this is

$$\phi(k, \mathbf{r}_i) = \boldsymbol{\psi}^t(k, \mathbf{r}_i) \cdot \mathbf{a} \tag{5.9}$$

$$\phi(k, \mathbf{r}_j) = Rg\boldsymbol{\psi}^t(k, \mathbf{r}_j) \cdot \mathbf{b} \tag{5.10}$$

$$\mathbf{b} = \boldsymbol{\alpha}_{ji}\mathbf{a} \tag{5.11}$$

The analogous matrix notation for vector waves is

$$\mathbf{E}(k, \mathbf{r}_i) = \begin{bmatrix} \mathbf{M}^t(k, \mathbf{r}_i) & \mathbf{N}^t(k, \mathbf{r}_i) \end{bmatrix} \begin{bmatrix} \mathbf{a} \\ \mathbf{b} \end{bmatrix} \tag{5.12}$$

$$\mathbf{E}(k, \mathbf{r}_j) = \begin{bmatrix} \Re\mathbf{M}^t(k, \mathbf{r}_j) & \Re\mathbf{N}^t(k, \mathbf{r}_j) \end{bmatrix} \begin{bmatrix} \mathbf{c} \\ \mathbf{d} \end{bmatrix} \tag{5.13}$$

$$\begin{bmatrix} \mathbf{c} \\ \mathbf{d} \end{bmatrix} = \begin{bmatrix} \mathbf{A}_{ji} & \mathbf{B}_{ji} \\ \mathbf{B}_{ji} & \mathbf{A}_{ji} \end{bmatrix} \begin{bmatrix} \mathbf{a} \\ \mathbf{b} \end{bmatrix} \tag{5.14}$$

In both scalar and vector cases the transpose, ^{t}, means align the harmonic indices along columns. Technically the sums are infinite, but the matrices are always truncated in practice.

5.2 Diagonalization

Explicit expressions for the matrix elements of the scalar and vector translation matrices are given in [2], but are given in terms of Wiger 3-j symbols, which are slow and inaccurate to compute. Fast recursions for quickly computing the scalar and vector translation matrices were developed in [22, 23, 24].

The standard way to compute the translation matrix is to break it into three matrices. The local frame, i, is first rotated to point its z-axis toward the destination frame, j. The local frame is then axially translated. Finally, the frame is then rotated back. This effectively diagonalizes the translation matrix. Specifically, if the vector $\mathbf{r}_{ji}$ points from the origin of frame i to the origin of frame j, with magnitude r_{ji} and spherical angles θ_{ji}, ϕ_{ji}, shown in Figure 5.1, then the scalar translation matrix is expanded (diagonalized) as

$$\boldsymbol{\alpha}_{ji} = \mathbf{D}^*(\phi_{ji} + \pi/2, \theta_{ji}, 0) \cdot \boldsymbol{\alpha}_z(r_{ji}) \cdot \mathbf{D}(\phi_{ji} + \pi/2, \theta_{ji}, 0) \tag{5.15}$$

where $\mathbf{D}$ is the rotation matrix with ZXZ convention, and $\boldsymbol{\alpha}_z(r_{ji})$ is the axial translation matrix in the $+z$ direction. The factor of $\pi/2$ ensures that θ_{ji} tips the z-axis toward the destination frame with a right-handed X rotation. The sequence starts with an inverse rotation on the right size because we first view the field, which is fixed in the global frame, from the point of view of the rotated frame. Next, this point of view is translated along the z-axis of the rotated frame. Finally, the reverse rotation (in this case a forward rotation) is applied to bring the frame back to be parallel with the global frame. This procedure also applies to the vector translation matrices $\mathbf{A}_{ji}$ and $\mathbf{B}_{ji}$.

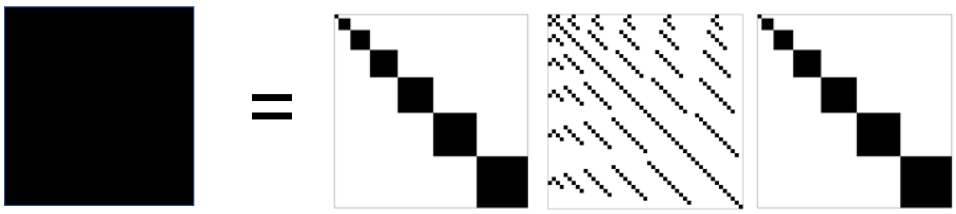


Figure 5.2: Diagonalization of the scalar translation matrix. L = 6. Left is the full translation matrix. Right shows the two rotation matrices bookending the z-axial translation matrix.

While one can reduce the recurrence relations for the full translation matrices given in [23, 24] to the axial case, equations for the axial scalar translation are derived directly in [22], from which the axial vector translations follow. These are given succinctly in [25].

The reason for the dramatic speed up of the matrix-vector multiplication in the case of diagonalization over the full matrix is because the rotation matrix is block diagonal and the z-axial translation requires the fewest non-zero terms of all possible translation directions. Together these yield many fewer overall multiplications than when using the full matrix, and the computational advantage grows as the matrix grows. In addition, the rotation matrix only needs to be computed once (the inverse is just conjugate). If the translation matrix is non-square, as will be the case when the number of harmonics between two frames is different, then the rotation matrix can be computed once for the larger of L and L', which are the largest degree along row and column, respectively, and then truncated. Figure 5.2 illustrates the diagonalization of the translation matrix. The matrices must be applied individually to the expansion coefficients from right to left, otherwise the benefits of diagonalization are lost.

5.3 Scalar Translation Matrix

In this section we give a routine for the scalar axial translation matrix. We then derive a routine for indexing and computing the inherently sparse scalar axial translation matrix on a 1D array. We also give a routine that accomplishes the diagonalized matrix-vector multiplication and operates on the expansion coefficients directly. Finally, we provide a routine that computes the full scalar translation matrix directly.

5.3.1 Scalar Axial Translation Matrix

The recurrence equations for the scalar axial translation matrix, (5.15), follow [22]. The matrix elements are zero except for $m = m'$. Let (l, m) and (l', m') correspond to the rows and columns, respectively. While the sums are technically infinite, they are always truncated at maximum orders L and L', respectively.

The calculation is initialized with

$$\alpha_{l,0,0,0} = (-1)^l \sqrt{2l+1} h_l^{(1)}(kr_{ji}) \tag{5.16}$$

Next, $\alpha_{l,m,l',|l'|}$ is computed from

$$\alpha_{l,l'+1,l'+1,l'+1} = \sqrt{\frac{2l'+3}{2(l'+1)}} \left[\sqrt{\frac{(l+l'+1)(l+l')}{(2l-1)(2l+1)}} \alpha_{l-1,l',l',l'} + \sqrt{\frac{(l-l'+1)(l-l')}{(2l+3)(2l+1)}} \alpha_{l+1,l',l',l'} \right] \tag{5.17}$$

and $\alpha_{l,-l',l',-l'} = \alpha_{l,l',l',l'}$. The remaining coefficients are obtained for $m = \pm l'$ using

$$\alpha_{l,m,l'+1,m} = \sqrt{2l'+3} \left[\sqrt{\frac{(l+l')(l-l')}{(2l-1)(2l+1)}} \alpha_{l-1,m,l',m} - \sqrt{\frac{(l+l'+1)(l-l'+1)}{(2l+3)(2l+1)}} \alpha_{l+1,m,l',m} \right] \tag{5.18}$$

and $m \neq \pm l'$ using

$$\begin{aligned} \alpha_{l,m,l'+1,m} \quad = \quad & \sqrt{\frac{(2l'+3)(2l'+1)}{(l'+m+1)(l'-m+1)}} \left[\sqrt{\frac{(l'+m)(l'-m)}{(2l'+1)(2l'-1)}} \alpha_{l,m,l'-1,m} \right. \\ & \left. + \sqrt{\frac{(l+m)(l-m)}{(2l+1)(2l-1)}} \alpha_{l-1,m,l',m} - \sqrt{\frac{(l+m+1)(l-m+1)}{(2l+3)(2l+1)}} \alpha_{l+1,m,l',m} \right] \end{aligned} \tag{5.19}$$

That's about as confusing as it gets. Some comments on computation:

1. The equations in [22] allow for translation in $\pm z$ directions. We only translate in $+z$.

2. The algorithm iterates on $l + 1$ for an element at $l' + 1$. This has a cascading effect that, in order to compute the lower right block at $l = L$ and $l' = L'$, an additional element at $L + 1$ is required, and so on. For example, if the translation matrix is square, we must fill a triangular region to $l = L + L'$ at $l' = 0$. This is explained in detail in [23]. Once the computation is finished, the matrix is cropped to the intended size.

3. We need to accommodate non-square matrices for translation of expansions of different harmonic content. Because of the filling requirement above, it is advantageous to create the fill region in the direction of the larger of L and L'. Rather than rewrite the routine to fill along rows or columns, we can make use of the follow transpose relation

$$\alpha^{ij}_{lm,l'm'} = (-1)^{l+l'} \alpha^{ji}_{lm,l'm'} \tag{5.20}$$

   The triangular fill region is appended to the bottom of the matrix, along the rows up to $\max(L, L')$. The matrix is then cropped and transposed if $L' > L$.

4. The computation is the same for α or β, only the Bessel function changes at the initialization.

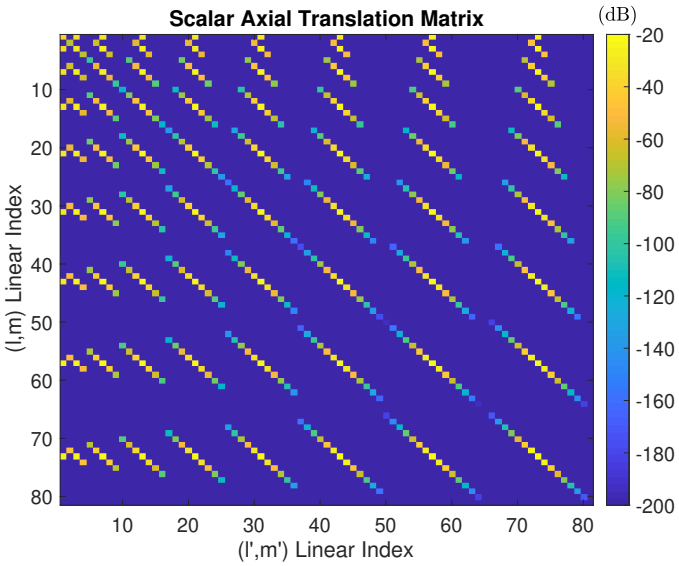


Figure 5.3: Scalar axial translation matrix, $L = L' = 8$, $k = 1$, $r_{ji} = 100$.

The routine `alphaz` takes as input the largest row degree, L, the largest column degree L', wavenumber k, axial translation distance r_{ji}, and returns the translation matrix in full, including zeros. Use string switch `'rg'` for the regular form. This routine is a good starting point for visualization, but the output matrix should not be used as is for matrix-vector multiplication due to the zeros, or it should be converted to a sparse matrix. The routine is written with the variable n in place of l' for clarity.

```
function [alpha] = alphaz(L,Lp,k,rji,rgstr)
% Scalar axial translation matrix
%
% L,Lp:     Maximum row, column degree harmonic
% k:        Background wavenumber
% rji:      Translation distance along +z
% rgstr:    'rg' for beta = Rg{alpha}, uses spherical Bessel j_l(kr)
%
% alpha:    Translation matrix of size N1xN2
%           N1 = L^2 + 2*L + 1
%           N2 = Lp^2 + 2*Lp + 1
%
% Dependencies: lm2ind, sbesselh, sbesselj

% Determine maximum order and transpose condition
flip = 0;
if Lp > L
    tmp = Lp;
    Lp = L;
    L = tmp;
    flip = 1;
end
L3 = L + Lp;
tot1 = L^2 + 2*L + 1;
tot2 = Lp^2 + 2*Lp + 1;
tot3 = L3^2 + 2*L3 + 1;
alpha = zeros(tot3,tot2);
str = 'mono';

% Initialize, alpha_{l0;00}
nk = lm2ind(0,0,str);
for l=0:L3,
    lm = lm2ind(l,0,str);
    if nargin == 5 && strcmp(rgstr,'rg')
        bess = sbesselj(l,k*rji);
    else
```

```
        bess = sbesselh(l,k*rji);
    end
    alpha(lm,nk) = (-1)^l*sqrt(2*l+1)*bess;
end

% alpha_{l,n+1;n+1,n+1}
for n = 0:(Lp-1),
for l = (n+1):(L3-n-1),
    lnp1 = lm2ind(l,n+1,str);
    np1np1 = lm2ind(n+1,n+1,str);
    nn = lm2ind(n,n,str);
    lp1n = lm2ind(l+1,n,str);
    if l-1 >= 0
        lm1n = lm2ind(l-1,n,str);
        c2 = sqrt((l+n+1)*(l+n)/(2*l-1)/(2*l+1));
        c2 = c2*alpha(lm1n,nn);
    else
        c2 = 0;
    end
    c1 = sqrt((2*n+3)/(2*(n+1)));
    c3 = sqrt((l-n+1)*(l-n)/(2*l+3)/(2*l+1));
    alpha(lnp1,np1np1) = c1*(c2 + c3*alpha(lp1n,nn));
end
end

% alpha_{l,-n;-n,-n}
for n = 1:Lp,
for l = n:(L3-n),
    ln = lm2ind(l,n,str);
    nn = lm2ind(n,n,str);
    lmn = lm2ind(l,-n,str);
    nmn = lm2ind(n,-n,str);
    alpha(lmn,nmn) = alpha(ln,nn);
end
end

% alpha_{lm,n+1,m}, m = +/-n
for n=0:(Lp-1),
for l=0:(L3-n-1),
    lim = min(l,n);
    for m=[-lim,lim],
        lm = lm2ind(l,m,str);
        np1m = lm2ind(n+1,m,str);
        nm = lm2ind(n,m,str);
        lp1m = lm2ind(l+1,m,str);
        c1 = sqrt(2*n+3);
        if abs(m) <= l-1
            lm1m = lm2ind(l-1,m,str);
            c2 = sqrt((l+n)*(l-n)/(2*l-1)/(2*l+1));
            c2 = c2*alpha(lm1m,nm);
        else
            c2 = 0;
        end
        c3 = sqrt((l+n+1)*(l-n+1)/(2*l+3)/(2*l+1));
        alpha(lm,np1m) = c1*(c2 - c3*alpha(lp1m,nm));
    end
end
end

% alpha_{lm,n+1,m}, m ~= +/-n
for n=1:(Lp-1),
for l=0:(L3-n-1),
    lim = min(l,n-1);
    for m=-lim:lim,
        lm = lm2ind(l,m,str);
        np1m = lm2ind(n+1,m,str);
        nm1m = lm2ind(n-1,m,str);
        nm = lm2ind(n,m,str);
```

```
        lp1m = lm2ind(l+1,m,str);
        c0 = sqrt((n+m+1)*(n-m+1)/(2*n+1)/(2*n+3));
        c1 = sqrt((n+m)*(n-m)/(2*n+1)/(2*n-1));
        if abs(m) <= l-1
            lm1m = lm2ind(l-1,m,str);
            c2 = sqrt((l+m)*(l-m)/(2*l+1)/(2*l-1));
            c2 = c2*alpha(lm1m,nm);
        else
            c2 = 0;
        end
        c3 = sqrt((l+m+1)*(l-m+1)/(2*l+3)/(2*l+1));
        alpha(lm,np1m) = (1/c0)*(c1*alpha(lm,nm1m) + c2 - c3*alpha(lp1m,nm));
    end
end
end

% trim the matrix
alpha = alpha(1:tot1,:);

% apply (-1)^(l+n) if needed
if flip
    for n=0:Lp,
    for l=0:L,
        if mod(n+l,2) == 1
            lim = min(l,n);
            for m=-lim:lim,
                lm = lm2ind(l,m,str);
                nm = lm2ind(n,m,str);
                alpha(lm,nm) = (-1)^(n+l)*alpha(lm,nm);
            end
        end
    end
    end
    alpha = alpha.';
end
```

5.3.2 Sparse Scalar Axial Translation Matrix

We want to compute the inherently sparse axial translation matrix on a 1D array, to avoid preallocating a potentially massive, mostly zero, matrix. Indexing is more complicated than for the sparse rotation matrix due to different sized diagonal subblocks as well as the triangular fill region requirement. Analytic formulas for sparse indexing are derived and used below. As implemented, these add a little computation time relative to indexing on the full preallocated matrix.

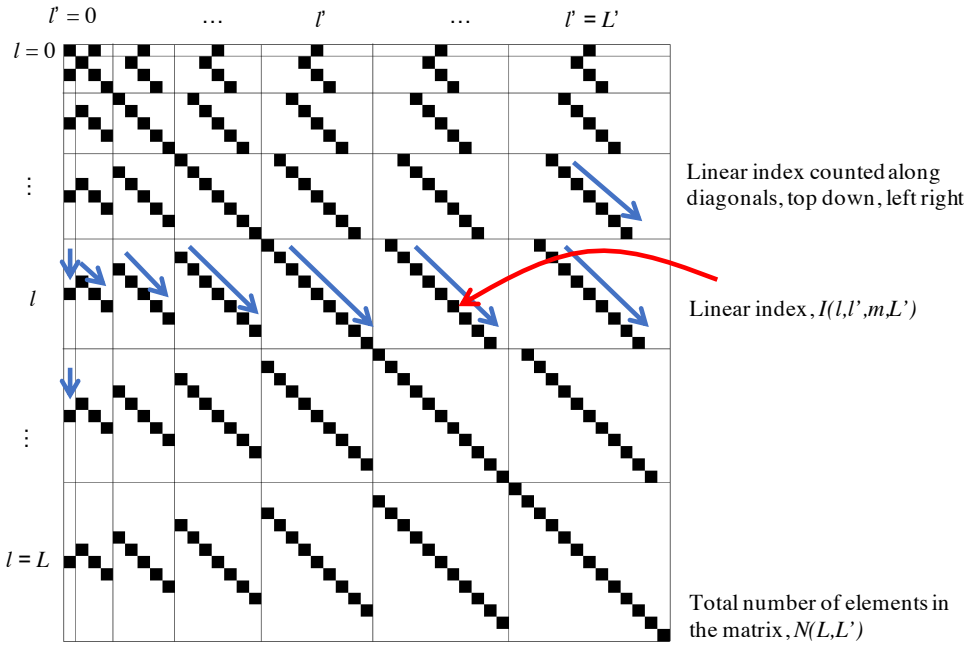


Figure 5.4: Indexing the diagonal subblocks of the sparse scalar axial translation matrix. Blue arrows indicate the direction of the linear indexing.

The total number of non-zero matrix elements in the scalar axial translation matrix, but not including the required triangular fill region, is counted as

$$N(L,L') = \sum_{l'=0}^{L'} \sum_{l=0}^{L} \sum_{m=-\min(l,l')}^{\min(l,l')} 1 \tag{5.21}$$

$$= \sum_{l'=0}^{L'} \sum_{l=0}^{L} (2\,\min(l,l') + 1) \tag{5.22}$$

This counts the total number of elements in a matrix that has $L \times L'$ subblocks. The number of non-zero elements along the diagonal of a subblock is determined by the smaller of the l and l'. See Figure 5.4. Plugging (5.22) into Wolfram Alpha we get

$$N(L,L') = \begin{cases} 1 & L = L' = 0 \\ L'+1 & L = 0, L' > 0 \\ L+1 & L' = 0, L > 0 \\ -L'^3/3 + L'^2 L + L'(2L + 4/3) + L + 1 & L' < L, L > 0, L' > 0 \\ -L^3/3 + L^2 L' + L(2L' + 4/3) + L' + 1 & L' > L, L > 0, L' > 0 \\ 2/3L^3 + 2L^2 + 7/3L + 1 & L' = L, L > 0, L' > 0 \end{cases} \tag{5.23}$$

Next, the total number of nonzero elements in the triangular fill region, see Figure 5.5, assuming $L \geq L'$, and assuming that the original matrix is extend along rows, is given by

$$N_{\text{fill}}(L, L') = \sum_{l'=0}^{L'-1} \sum_{l=L+1}^{L+L'-l'} (2l' + 1) = L'(2L'^2 + 3L' + 1)/6 \tag{5.24}$$

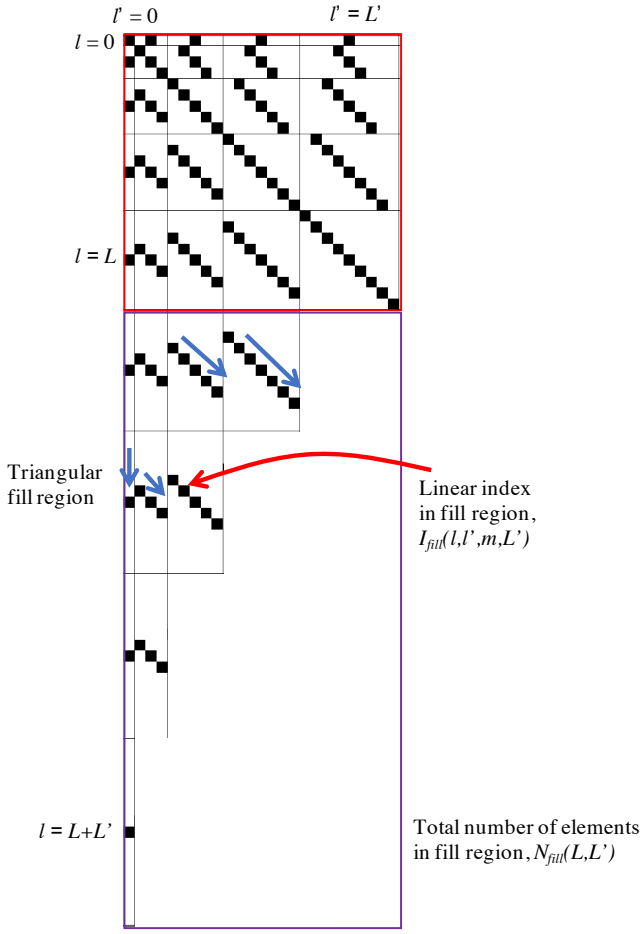


Figure 5.5: Indexing the diagonal subblocks of the triangular fill region of the sparse scalar axial translation matrix. Blue arrows indicate the direction of the linear indexing.

Because we have defined the fill region as additional matrix rows, the 1D crop is easiest if we index the diagonal (l, l') subblocks as "row"-major (left-right, top-down), and where we index the diagonals of the subblocks from upper left to lower right. See again Figure 5.4. Once the computation is complete we keep the first $N(L, L')$ elements and apply the transpose if $L' > L$.

The "row"-major 1D linear index for the axial translation matrix, but not including the triangular fill region, is given by

$$I(l, l', m, L') = N(l - 1, L') + \sum_{i=0}^{l'-1} (2\min(l, i) + 1) + (\min(l, l') + m + 1) \tag{5.25}$$

where

$$\sum_{i=0}^{l'-1} (2\min(l, i) + 1) = \left\{ \begin{array}{cc} l'^2, & l' = 1 \vee (l + 1 > l' \wedge l' > 1) \\ -l^2 - l + 2ll' + l' & \text{otherwise} \end{array} \right\} \tag{5.26}$$

$N(l-1,L')$ counts the submatrix containing all diagonal subblocks up to but not including the desired l row of diagonal blocks. The second term counts completed column blocks in the current row block up to but not including the desired l'. The last term counts the desired diagonal. This counting requires knowledge of L'.

Finally, the linear index of an element in the triangular fill region under the conditions that $L \geq L'$, $l > L$, and $0 \leq l' \leq L'-1$ is

$$\begin{aligned} I_{\text{fill}}(l,l',m,L,L') &= N(L,L') + \sum_{i=L+1}^{l-1} \left(\sum_{j=0}^{L'-1-(i-L-1)} (2j+1) \right) \\ &\quad + \left(\sum_{j=0}^{l'-1} (2j+1) \right) + (l'+m+1) \qquad (5.27) \\ &= N(L,L') + \big(1/6(l-L-1)(2l^2 - l(4L+6Lp+7) + 2L^2 \\ &\quad + L(6Lp+7) + 6(Lp+1)^2)\big) + l'^2 + (l'+m+1) \qquad (5.28) \end{aligned}$$

The first term counts the number of elements in the actual matrix. The second term counts the completed row blocks up to but not including the current row block in the fill triangle. The third term counts the column blocks in the current row block up to but not including the active diagonal (made simpler by that fact that $l' < l$ in the fill region). The last term indexes the active diagonal.

The routine `alphazSparse` returns three column arrays of the row index, column index, and matrix entries of the sparse axial translation matrix. It is the same as `alphaz` but computed and output on a 1D array. Helper functions are given in Table 5.1. `Nalphaz` and `Nalphazfill` count the number of nonzero elements in the matrix and fill region, respectively. `indAlphazFill` is used to index the sparse matrix including the fill regions; it calls `indAlphaz` which indexes the primary sparse matrix. When $L' > L$, the transpose is accomplished by applying (5.20) and reordering row, column, and array elements to be consistent with the row-major indexing of `indAlphaz`.

Table 5.1: `alphazSparse` Helper Functions

Routine	Equation	Description
`Nalphaz`	(5.23)	Number of non-zero elements in $L \times L'$ sparse $\alpha_{lm,l'm'}$
`Nalphazfill`	(5.24)	Number of non-zero elements in fill region
`indAlphaz`	(5.25)	Row-major 1D linear index in sparse $\alpha_{lm,l'm'}$
`indAlphazFill`	(5.27)	Row-major 1D linear index in sparse fill region

```
function [row col alpha] = alphazSparse(L,Lp,k,rji,rgstr)
% Scalar axial translation matrix computed as sparse matrix
%
% L,Lp:     Maximum row, column degree harmonic
% k:        Background wavenumber
% rji:      Translation distance along +z
% rgstr:    'rg' for beta = Rg{alpha}, uses spherical Bessel j_l(kr)
%
% row,col:  row and column indices of sparse matrix entries
% alpha:    sparse matrix entries
%
% Dependencies: lm2ind, sbesselh, sbesselj,
%                   Nalphaz, Nalphazfill, indAlphazFill, indAlphaz

% Determine maximum degree and transpose condition
flip = 0;
if Lp > L
```

```
    tmp = Lp;
    Lp = L;
    L = tmp;
    flip = 1;
end
L3 = L + Lp;
str = 'mono';
NLLP = Nalphaz(L,Lp);
NN = NLLP + Nalphazfill(L,Lp);
alpha = zeros(NN,1);
row = zeros(NN,1);
col = zeros(NN,1);

% Initialize, alpha_{l0;00}
nk = lm2ind(0,0,str);
for l=0:L3,
   lm = lm2ind(l,0,str);
    if nargin == 5 && strcmp(rgstr,'rg')
        bess = sbesselj(l,k*rji);
    else
        bess = sbesselh(l,k*rji);
    end
    inds = indAlphazFill(l,0,0,L,Lp);
    row(inds) = lm;
    col(inds) = nk;
    alpha(inds) = (-1)^l*sqrt(2*l+1)*bess;
end

% alpha_{l,n+1;n+1,n+1}
for n = 0:(Lp-1),
for l = (n+1):(L3-n-1),
    lnp1 = lm2ind(l,n+1,str);
    np1np1 = lm2ind(n+1,n+1,str);
    if l-1 >= 0
        c2 = sqrt((l+n+1)*(l+n)/(2*l-1)/(2*l+1));
        c2 = c2*alpha(indAlphazFill(l-1,n,n,L,Lp));
    else
        c2 = 0;
    end
    c1 = sqrt((2*n+3)/(2*(n+1)));
    c3 = sqrt((l-n+1)*(l-n)/(2*l+3)/(2*l+1));
    inds = indAlphazFill(l,n+1,n+1,L,Lp);
    row(inds) = lnp1;
    col(inds) = np1np1;
    alpha(inds) = c1*(c2 + c3*alpha(indAlphazFill(l+1,n,n,L,Lp)));
end
end

% alpha_{l,-n;n,-n}
for n = 1:Lp,
for l = n:(L3-n),
    lmn = lm2ind(l,-n,str);
    nmn = lm2ind(n,-n,str);
    inds = indAlphazFill(l,n,-n,L,Lp);
    row(inds) = lmn;
    col(inds) = nmn;
    alpha(inds) = alpha(indAlphazFill(l,n,n,L,Lp));
end
end

% alpha_{lm,n+1,m}, m = +/-n
for n=0:(Lp-1),
for l=0:(L3-n-1),
    lim = min(l,n);
    for m=[-lim,lim],
        lm = lm2ind(l,m,str);
        np1m = lm2ind(n+1,m,str);
        c1 = sqrt(2*n+3);
```

```
        if abs(m) <= l-1
            c2 = sqrt((l+n)*(l-n)/(2*l-1)/(2*l+1));
            c2 = c2*alpha(indAlphazFill(l-1,n,m,L,Lp));
        else
            c2 = 0;
        end
        c3 = sqrt((l+n+1)*(l-n+1)/(2*l+3)/(2*l+1));
        inds = indAlphazFill(l,n+1,m,L,Lp);
        row(inds) = lm;
        col(inds) = np1m;
        alpha(inds) = c1*(c2 - c3*alpha(indAlphazFill(l+1,n,m,L,Lp)));
    end
end
end

% alpha_{lm,n+1,m}, m ~= +/-n
for n=1:(Lp-1),
for l=0:(L3-n-1),
    lim = min(l,n-1);
    for m=-lim:lim,
        lm = lm2ind(l,m,str);
        np1m = lm2ind(n+1,m,str);
        c0 = sqrt((n+m+1)*(n-m+1)/(2*n+1)/(2*n+3));
        c1 = sqrt((n+m)*(n-m)/(2*n+1)/(2*n-1));
        if abs(m) <= l-1
            c2 = sqrt((l+m)*(l-m)/(2*l+1)/(2*l-1));
            c2 = c2*alpha(indAlphazFill(l-1,n,m,L,Lp));
        else
            c2 = 0;
        end
        c3 = sqrt((l+m+1)*(l-m+1)/(2*l+3)/(2*l+1));
        inds = indAlphazFill(l,n+1,m,L,Lp);
        row(inds) = lm;
        col(inds) = np1m;
        alpha(inds) = (1/c0)*(c1*alpha(indAlphazFill(l,n-1,m,L,Lp)) + c2 ...
                        - c3*alpha(indAlphazFill(l+1,n,m,L,Lp)));
    end
end
end

% trim the array
alpha = alpha(1:NLLP);
row = row(1:NLLP);
col = col(1:NLLP);

% if transpose, apply (-1)^(l+n) and reorder
% to be consistent with row-major indexing from indAlphaz
if flip
    rtmp = zeros(NLLP,1);
    ctmp = rtmp;
    atmp = rtmp;
    for n=0:Lp,
    for l=0:L,
        lim = min(l,n);
        for m=-lim:lim,
            ind1 = indAlphaz(l,n,m,Lp);
            ind2 = indAlphaz(n,l,m,L);
            atmp(ind2) = (-1)^(n+l)*alpha(ind1);
            ctmp(ind2) = row(ind1);
            rtmp(ind2) = col(ind1);
        end
    end
    end
    alpha = atmp;
    row = rtmp;
    col = ctmp;
end
```

```
function Nl = Nalphaz(L,Lp)
% Number of non-zero elements in sparse scalar axial translation matrix
%
% L:    Maximum row order l=0:L
% Lp:   Maximum column order lp=0:Lp
%
% Nl:   Number of nonzero elements

if L == 0 && Lp == 0
    Nl = 1;
elseif L == 0 && (Lp>0)
    Nl = Lp + 1;
elseif Lp == 0 && (L>0)
    Nl = L + 1;
elseif (L>0) && (Lp>0) && (Lp<L)
    Nl = -Lp^3/3 + Lp^2*L + Lp*(2*L + 4/3) + L + 1;
elseif (L>0) && (Lp>0) && (Lp>L)
    Nl = -L^3/3 + L^2*Lp + L*(2*Lp + 4/3) + Lp + 1;
elseif (L>0) && (Lp>0) && (Lp==L)
    Nl = 2/3*L^3 + 2*L^2 + 7/3*L + 1;
else
    disp('bad index')
    return
end
Nl = round(Nl);


function Nfill = Nalphazfill(L,Lp)
% Number of non-zero elements in the fill region of the
% sparse scalar axial translation matrix.  Assumes L >= Lp
%
% L:    Maximum row order in actual matrix, l=0:L
% Lp:   Maximum column order in actual matrix lp=0:Lp
%
% Nfill:   Number of nonzero elements in the fill region

if L < Lp
    disp('bad index')
    return
end
Nfill = 1/6*Lp*(2*Lp^2 + 3*Lp + 1);
Nfill = round(Nfill);


function [ind] = indAlphaz(l,lp,m,Lp)
% Row-major linear index of the sparse scalar axial translation matrix
%
% l,lp,m:   Harmonix index (l,lp,m)
% Lp:       Maximum column degree lp=0:Lp
%
% ind:      Sparse matrix linear index
%
% Dependencies: Nalaphz

if m > min(l,lp) || l < 0 || lp < 0
    disp('bad index')
else
    if l == 0
        ind1 = 0;
    else
        ind1 = Nalphaz(l-1,Lp);
    end
    if lp == 0
        ind2 = 0;
    elseif lp == 1 || ((l+1>lp) && (lp > 1))
```

```
        ind2 = lp^2;
    else
        ind2 = -l^2 - l + 2*l*lp + lp;
    end
    ind3 = min(l,lp) + m + 1;
    ind = ind1 + ind2 + ind3;
end
ind = round(ind);


function [ind] = indAlphazFill(l,lp,m,L,Lp)
% Row-major linear index for the sparse scalar axial translation matrix
% including the fill region.  Assumes that the fill region is added along
% rows and L >= Lp.
%
% l,lp,m:   Harmonic index (l,lp,m)
% L:        Maximum row degree l=0:L
% Lp:       Maximum column degree lp=0:Lp
%
% ind:      Sparse matrix linear index including the fill region
%
% Dependencies: Nalaphz, indAlphaz

if m > min(l,lp) || l < 0 || lp < 0 || Lp > L
    disp('bad index')
    return
elseif l <= L && lp <= Lp
    ind = indAlphaz(l,lp,m,Lp);
elseif l > L
    ind = Nalphaz(L,Lp) + lp^2 + lp + m + 1;
    if l > L + 1
        ind = ind + 1/6*(l-L-1)*(2*l^2-l*(4*L+6*Lp+7) ...
            +2*L^2+L*(6*Lp+7)+6*(Lp+1)^2);
    end
else
    disp('bad index')
    return
end
```

5.3.3 Diagonalized Scalar Translation Matrix

After creating sparse rotation and axial translation matrices, the matrix-vector multiplication should be carried out right to left in sequence otherwise the benefit of diagonalization is lost. For example, if `a` is a vector of expansion coefficients, and the matrices are in Matlab's sparse matrix format, then the multiplication should be carried out as follows `b = conj(D)*(Alphaz*(D*a))`.

The routine `scalarTranslation` takes as input a vector of expansion coefficients in frame i, $\mathbf{a}$, the maximum degrees L and L' of the translation matrix and the translation vector $\mathbf{r}_{ji}$ and returns expansion coefficients in frame j, $\mathbf{b}$. The length of $\mathbf{a}$ should be $L'^2 + 2L' + 1$. Use `'rg'` for the regular form of the translation. This uses the routines for the sparse forms of the rotation matrix and axial translation matrix to save memory. If the entire matrix is desired then include the optional second output.

```
function [blm alp] = scalarTranslation(alm,L,Lp,k,xji,rgstr)
% Scalar translation matrix
%
% alm:      Expansion coefficients in frame i
% L,Lp:     Maximum row/col degrees of the translation matrix
% rji:      3x1 Cartesian vector the points from frame i to frame j
% rgstr:    Use 'rg' for regular form of translation matrix
%
% blm:      Expansion coefficients in frame j
% alp:      [optional] entire translation matrix
%
% Dependencies: cart2sph,alphazSparse,DlmpSparse,NDlmpSparse

NL = L^2 + 2*L + 1;
NLp = Lp^2 + 2*Lp + 1;
Lmax = max([L Lp]);
Lmin = min([L Lp]);
NLmax = max([NL NLp]);
NLmin = min([NL NLp]);
alm = alm(:);
if length(alm) ~= NLp
    error('alm array length does not match Lp')
end

% convert translation vector to spherical
[rji thetaji phiji] = cart2sph(xji(1),xji(2),xji(3));

% compute sparse axial translation
[row col alpz] = alphazSparse(L,Lp,k,rji,rgstr);
alpz = sparse(row,col,alpz,NL,NLp);

% compute rotation matrix for larger of L and Lp
[row col D] = DlmpSparse(Lmax,phiji+pi/2,thetaji,0,'mono');
D1 = sparse(row,col,D,NLmax,NLmax);

% grab the matrix entries for the lesser of L and Lp
Ngrab = NDlmpSparse(Lmin,'mono');
D2 = sparse(row(1:Ngrab),col(1:Ngrab),D(1:Ngrab),NLmin,NLmin);

% mutliply depending on L and Lp
if Lp >= L
    blm = D2'*(alpz*(D1*alm));
else
    blm = D1'*(alpz*(D2*alm));
end

if nargout == 2
    if Lp >= L
        alp = D2'*(alpz*(D1));
    else
        alp = D1'*(alpz*(D2));
    end
end
```

5.3.4 Full Scalar Translation Matrix

Recursion relations to compute the full scalar translation matrix were derived in [23]. While diagonalization is efficient for large matrices, using the full-matrix recursion is more efficient when the number of rows or columns in the matrix are small. The sums of the additional theorem are technically infinite, however, if the field being translated is known to have low harmonic content, then there is no need to compute the matrix beyond the maximum harmonic. For example, the field expansion of a pure scalar dipole only contains harmonics up to degree $l = 1$ (i.e., $(l, m) = (0,0),(1,-1),(1,0),(1,1)$). Translating this field as incoming waves in the frame of a large scattering object requires only four matrix columns, but a large number rows. The same computation done with diagonalization needs a square rotation matrix to match the number of harmonics of the rows, which is more computation than necessary. Finally, being able to compute the full matrix is useful for cross checking.

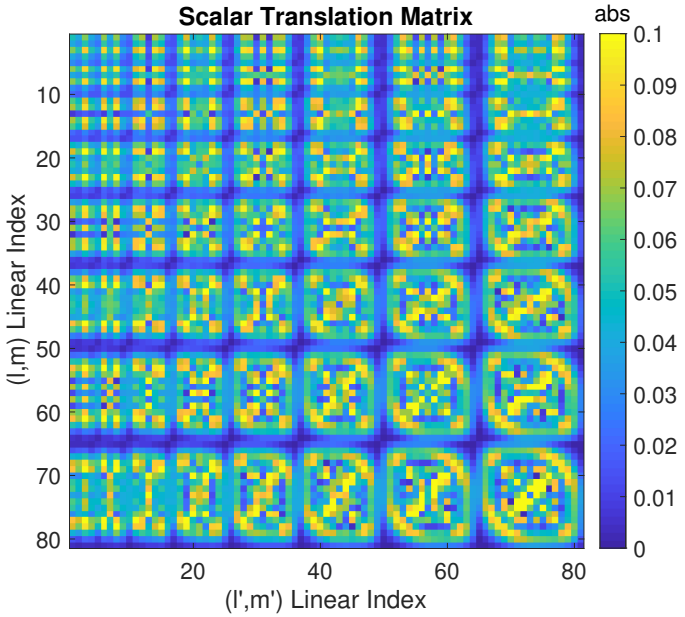


Figure 5.6: Scalar translation matrix, $L = L' = 8$, $k = 1$, $r_{ji} = [12, 5, 15]$.

Adopting the notation from [23], which uses the pairings $(\nu\mu, nm)$ for our $(lm, l'm')$, the recurrence relations for the full scalar translation matrix are initialized with

$$\alpha_{\nu\mu,00} = \sqrt{4\pi}(-1)^{\nu} Y^*_{\nu\mu}(\theta, \phi) h^{(1)}_{\nu}(kr) \tag{5.29}$$

where (r, θ, ϕ) are the spherical coordinates of the vector, $\mathbf{r}_{ji}$, that points from the originating frame to the translated frame. The first recursion relation is

$$b^+_{nn}\alpha_{\nu\mu,n+1,n+1} = b^+_{\nu-1,\mu-1}\alpha_{\nu-1,\mu-1,nn} + b^-_{\nu+1,\mu-1}\alpha_{\nu+1,\mu-1,nn} \tag{5.30}$$

This is also used to iterated on $b^+_{n,-n}\alpha_{\nu\mu,n+1,-(n+1)}$. The second relation is

$$a^+_{nm}\alpha_{\nu\mu,n+1,m} = -a^-_{nm}\alpha_{\nu\mu,n-1,m} + a^+_{\nu-1,\mu}\alpha_{\nu-1,\mu,nm} + a^-_{\nu+1,\mu}\alpha_{\nu+1,\mu,nm} \tag{5.31}$$

The coefficients are

$$a^+_{nm} = -\sqrt{\frac{(n+m+1)(n-m+1)}{(2n+1)(2n+3)}} \tag{5.32}$$

$$a^-_{nm} = \sqrt{\frac{(n+m)(n-m)}{(2n+1)(2n-1)}} \tag{5.33}$$

$$b_{nm}^{+} = \sqrt{\frac{(n+m+2)(n+m+1)}{(2n+1)(2n+3)}} \tag{5.34}$$

$$b_{nm}^{-} = \sqrt{\frac{(n-m)(n-m-1)}{(2n+1)(2n-1)}} \tag{5.35}$$

Like the axial translation matrix, the computation requires a triangular fill region. As before, the fill region is placed along the dimension of the matrix with the highest degree harmonic, because we only need to fill to the smaller of L and L'. We always fill along rows, so if the matrix needs to be flipped, the following transpose relation is used

$$\alpha_{mn,\nu\mu} = (-1)^{\nu+\mu+m+n}\alpha_{\nu,-\mu,n,-m} \tag{5.36}$$

The routine `alpha` computes the full scalar translation matrix using the recursion relations above. It takes as input the largest row degree, L, the largest column degree L', wavenumber k, axial translation vector $\mathbf{r}_{ji}$, and returns the translation matrix linearly indexed along rows and columns. Use string switch `'rg'` for the regular form. The computation fills along rows for the larger of L and L', then transposes the matrix if necessary. This routine matches the outputs of the diagonalized version to machine precision.

```
function [alpha] = alpha(L,Lp,k,rji,rgstr)
% 3D scalar translation matrix, alpha

% L,Lp:     Maximum degree in row and column
% k:        Background wave number
% rji:      [x,y,z] vector from the soruce to receiver frame
% rgstr:    'rg' for beta = Rg{alpha}, uses spherical Bessel j_l(kr)
%
% alpha:    Translation matrix of size N1xN2
%           N1 = L^2 + 2*L + 1
%           N2 = Lp^2 + 2*Lp + 1
%
% Dependencies: cart2sph, sbesselh, sbesselj, am, ap, bm, bp, bmm, bpm

% convert rji to spherical coordinates
[r th phi] = cart2sph(rji(1),rji(2),rji(3));

% Determine maximum order and transpose condition
flip = 0;
if Lp > L
    tmp = Lp;
    Lp = L;
    L = tmp;
    flip = 1;
end
L3 = L + Lp;
tot1 = L^2 + 2*L + 1;
tot2 = Lp^2 + 2*Lp + 1;
tot3 = L3^2 + 2*L3 + 1;
alpha = zeros(tot3,tot2);
str = 'mono';

% alpha_{vu,00}
ylm = sphericalY(L3,th,phi,'mono');
if nargin == 5 && strcmp(rgstr,'rg')
    bess = sbesselj(0:L3,k*r);
else
    bess = sbesselh(0:L3,k*r);
end
for v=0:L3,
for u=-v:v,
    vu = v^2 + v + u + 1;
    alpha(vu,1) = sqrt(4*pi)*(-1)^(v)*conj(ylm(vu))*bess(v+1);
end
end

% b_nn+ alpha_vu,n+1,n+1 = ...
```

```
for n = 0:(Lp-1),
    nn = n^2 + 2*n + 1;
    np = n+1;
    npnp = np^2 + 2*np + 1;
    bpnn = bp(n,n);
    for v = 0:(L3-1-n),
    for u = (-v):v,
        in1 = v^2 + v + u + 1;
        vp = v+1;
        vm = v-1;
        um = u-1;
        vu1 = vm^2 + vm + um + 1;
        vu2 = vp^2 + vp + um + 1;
        bp1 = bp(vm,um);
        bm1 = bm(vp,um);
        if vm < 0 || um < -vm
            alpha(in1,npnp) = (1/bpnn)*(bm1*alpha(vu2,nn));
        else
            alpha(in1,npnp) = (1/bpnn)*(bp1*alpha(vu1,nn) + bm1*alpha(vu2,nn));
        end
    end
    end
end

% b_n,-n+ alpha_vu,n+1,-(n+1) = ...
for n = 0:(Lp-1),
    nn = n^2 + 1;
    np = n+1;
    npnp = np^2 + 1;
    bpnn = bpm(n,-n);
    for v = 0:(L3-1-n),
    for u = (-v):v,
        in1 = v^2 + v + u + 1;
        vp = v+1;
        vm = v-1;
        up = u+1;
        vu1 = vm^2 + vm + up + 1;
        vu2 = vp^2 + vp + up + 1;
        bp1 = bpm(vm,up);
        bm1 = bmm(vp,up);
        if vm < 0
            alpha(in1,npnp) = (1/bpnn)*(bm1*alpha(vu2,nn));
        else
            alpha(in1,npnp) = (1/bpnn)*(bp1*alpha(vu1,nn) + bm1*alpha(vu2,nn));
        end
    end
    end
end

% a_nm+ alpha_vu,n+1,m = -a_nm- alpha_vu,n1,m ...
for n = 0:(Lp-1),
for m = -n:n,
    np = n+1;
    nm = n-1;
    nm0 = n^2 + n + m + 1;
    nmm = nm^2 + nm + m + 1;
    npm = np^2 + np + m + 1;
    ap1 = ap(n,m);
    am1 = am(n,m);
    for v = 0:(L3-n-1),
    for u = (-v):v,
        vp = v+1;
        vm = v-1;
        vu0 = v^2 + v + u + 1;
        vu1 = vm^2 + vm + u + 1;
        vu2 = vp^2 + vp + u + 1;
        ap2 = ap(vm,u);
        am2 = am(vp,u);
```

```
        if (m >= 0 && nm < 0) || (m<0 && nm <= 0)
            if vm < 0 || u < -vm
                alpha(vu0,npm) = (1/ap1)*(am2*alpha(vu2,nm0));
            else
                alpha(vu0,npm) = (1/ap1)*(ap2*alpha(vu1,nm0) + am2*alpha(vu2,nm0));
            end
        else
            if vm < 0 || u < -vm
                alpha(vu0,npm) = (1/ap1)*(-am1*alpha(vu0,nmm)+am2*alpha(vu2,nm0));
            else
                alpha(vu0,npm) = (1/ap1)*(-am1*alpha(vu0,nmm)+ap2*alpha(vu1,nm0) + am2*alpha(vu2,nm0));
            end
        end
    end
    end
end
end

% trim the matrix
alpha = alpha(1:tot1,:);

% apply (-1)^(n+v+m+u) if flip is needed
if flip
    alpha2 = zeros(tot1,tot2);
    for n=0:Lp,
    for m=-n:n,
        nm = n^2 + n + m + 1;
        nmm = n^2 + n - m + 1;
        for v=0:L,
        for u=-v:v,
            uv = v^2 + v + u + 1;
            uvm = v^2 + v - u + 1;
            alpha2(uv,nm) = (-1)^(n+v+m+u)*alpha(uvm,nmm);
        end
        end
    end
    end
    alpha = alpha2.';
end
end

% Helper functions
function amc = am(n,m)
    amc = sqrt((n+m)*(n-m)/(2*n+1)/(2*n-1));
end
function apc = ap(n,m)
    apc = -sqrt((n+m+1)*(n-m+1)/(2*n+1)/(2*n+3));
end
function bmc = bm(n,m)
    bmc = sqrt((n-m)*(n-m-1)/(2*n+1)/(2*n-1));
end
function bmc = bmm(n,m)
    bmc = -sqrt((n+m)*(n+m-1)/(2*n+1)/(2*n-1));
end
function bpc = bp(n,m)
    bpc = sqrt((n+m+2)*(n+m+1)/(2*n+1)/(2*n+3));
end
function bpc = bpm(n,m)
    bpc = -sqrt((n-m+2)*(n-m+1)/(2*n+1)/(2*n+3));
end
```

5.4 Vector Translation Matrix

This section gives routines for the vector axial translation matrix, sparse vector axial translation matrix, and an implementation of the diagonalized matrix-vector multiplication.

5.4.1 Vector Axial Translation Matrix

The vector axial translation matrices are found directly from the scalar axial translation matrix with the following relations:

$$\begin{aligned} A_{l,m,l',m} &= kr_{ji}\left[\frac{1}{(l+1)}\sqrt{\frac{(l+m+1)(l-m+1)}{(2l+3)(2l+1)}}\alpha_{l+1,m,l',m}\right. \\ &\quad \left. + \frac{1}{l}\sqrt{\frac{(l+m)(l-m)}{(2l+1)(2l-1)}}\alpha_{l-1,m,l',m}\right] + \alpha_{l,m,l',m} \end{aligned} \tag{5.37}$$

$$B_{l,m,l',m} = \frac{imkr_{ji}}{l(l+1)}\alpha_{l,m,l',m} \tag{5.38}$$

$$A_{l,m,l',m'} = B_{l,m,l',m'} = \alpha_{l,m,l',m'} = 0, \qquad m \neq m' \tag{5.39}$$

Because the first equation uses $l+1$, the scalar axial matrix needs one extra degree $l = L+1$. The equations in [22] appear to be missing several scaling factors, but [24] gives correct equations. Both [22, 24] derived translation matrices for partially normalized vector spherical waves functions. For fully normalized vector spherical wave functions, the vector translation matrices need to be modified as

$$\widetilde{A}_{l,m,l',m'} = \sqrt{\frac{l(l+1)}{l'(l'+1)}}A_{l,m,l',m'} \tag{5.40}$$

$$\widetilde{B}_{l,m,l',m'} = \sqrt{\frac{l(l+1)}{l'(l'+1)}}B_{l,m,l',m'} \tag{5.41}$$

The routine `AzBz` takes as input the maximum harmonic degrees L and L', wavenumber k, and magnitude of the z-axis translation, r_{ji}, and returns vector translation matrices $A_{l,m,l',m}$ and $B_{l,m,l',m}$. Use string switch `'rg'` for the regular form of the matrices, and `'norm'` for fully normalized vector wave functions.

```
function [A B] = AzBz(L,Lp,k,rji,rgstr,normstr)
% Vector axial translation matrix
%
% L,Lp:     Maximum row/column harmonic degree
% k:        Background wavenumber
% rji:      Translation distance along +z
% rgstr:    'rg' for beta = Rg{alpha}
% normstr:  'norm' for fully normalized wave functions
%
% A,B:      Translation matrices size N1 x N2
%           N1 = L^2 + 2*L
%           N2 = Lp^2 + 2*Lp
%
% Dependencies: lm2ind, alphaz

if nargin == 4
    rgstr = [];
    normstr = [];
end
% compute scalar axial translation
alpz = alphaz(L+1,Lp,k,rji,rgstr);

tot1 = L^2 + 2*L;
tot2 = Lp^2 + 2*Lp;
kr = k*rji;
A = zeros(tot1,tot2);
```

```
B = A;
for n=1:Lp,
for l=1:L,
    lim = min(l,n);
    for m=-lim:lim,
        lm = lm2ind(l,m,'mono');
        lp1m = lm2ind(l+1,m,'mono');
        nm = lm2ind(n,m,'mono');
        c1 = sqrt((l+m+1)*(l-m+1)/(2*l+1)/(2*l+3))/(l+1);
        c3 = m/(l*(l+1));
        if abs(m) <= l-1
            lmm1 = lm2ind(l-1,m,'mono');
            c2 = sqrt((l+m)*(l-m)/(2*l-1)/(2*l+1))/l;
            c2 = c2*alpz(lmm1,nm);
        else
            c2 = 0;
        end
        lm2 = lm2ind(l,m);
        nm2 = lm2ind(n,m);
        A(lm2,nm2) = kr*(c1*alpz(lp1m,nm) + c2) + alpz(lm,nm);
        B(lm2,nm2) = 1i*kr*c3*alpz(lm,nm);
    end
end
end
if nargin == 6 && strcmp(normstr,'norm')
    for n=1:Lp,
    for l=1:L,
        lim = min(l,n);
        for m=-lim:lim,
            const = sqrt(l*(l+1))/sqrt(n*(n+1));
            A(lm2ind(l,m),lm2ind(n,m)) = const*A(lm2ind(l,m),lm2ind(n,m));
            B(lm2ind(l,m),lm2ind(n,m)) = const*B(lm2ind(l,m),lm2ind(n,m));
        end
    end
    end
end
```

5.4.2 Sparse Vector Axial Translation Matrix

Because the vector axial translation matrices are computed directly from the scalar functions, they are sparse and can be computed on a 1D array. We only need to count the number of elements in the vector axial translation matrix and determine its sparse linear index. The number of elements in the sparse vector axial translation matrix is counted the same as the scalar case starting at $l = 1$ and $l' = 1$.

$$N(L,L') = \sum_{l'=1}^{L'}\sum_{l=1}^{L}\sum_{m=-\min(l,l')}^{\min(l,l')} 1 \tag{5.42}$$

$$= \sum_{l'=1}^{L'}\sum_{l=1}^{L}[2\min(l,l')+1] \tag{5.43}$$

Plugging this into Wolfram Alpha

$$N(L,L') = \begin{cases} 3 & L = L' = 1 \\ 3L & L' = 1, L > 1 \\ 3L' & L = 1, L' > 1 \\ -1/3L'^3 + LL'^2 + 2LL' + 1/3L' & L > 1, L' > 1, L > L' \\ -1/3L^3 + L'L^2 + 2L'L + 1/3L & L > 1, L' > 1, L' > L \\ L^2 + 5L + 2/3L'^3 + L'^2 - 14/3L' & L > 1, L' > 1, L = L' \end{cases} \tag{5.44}$$

Counting row-major, the sparse linear index is

$$I(l,l',m,L') = N(l-1,L') + \sum_{i=1}^{l'-1}(2\min(l,i)+1) + (\min(l,l')+m+1) \tag{5.45}$$

where

$$\sum_{i=1}^{l'-1}(2\min(l,i)+1) = \left\{ \begin{array}{cc} 3 & l+1 \geq l' \wedge ((l=1 \wedge l' \geq 2) \vee (l'=2 \wedge l>1)) \\ 2l(l'-2)+l'+1 & l+1 < l' \wedge ((l=1 \wedge l' \geq 2) \vee (l'=2 \wedge l>1)) \\ l'^2-1 & l>1 \wedge l+1 \geq l' \wedge l>2 \\ -l^2+l+(2l+1)(l'-1) & \text{otherwise} \end{array} \right\} \tag{5.46}$$

The routine `AzBzSparse` returns four column arrays of the row index, column index, and matrix entries of the sparse axial translation matrices. It is the same as `AzBz` but computed and output on a 1D array. The helper functions are `NAzBz.m` and `indAzBz.m`.

```
function [row col A B] = AzBzSparse(L,Lp,k,rji,rgstr,normstr)
% Vector axial translation matrix
%
% L,Lp:     Maximum row, column harmonic
% k:        Background wavenumber
% rji:      Translation distance along +z
% rgstr:    'rg' for beta = Rg{alpha}
% normstr:  'norm' for fully normalized wave functions
%
% row,col:  row and column indices of sparse matrix entries
% A, B:     sparse matrix entires
%
% Dependencies: lm2ind, sbesselh, sbesselj,
%               alphazSparse, NAzBz, indAzBz

if nargin == 4
    rgstr = [];
    normstr = [];
end
% sprase scalar axial translation matrix
[~, ~, alpz] = alphazSparse(L+1,Lp,k,rji,rgstr);
kr = k*rji;
A = zeros(NAzBz(L,Lp),1);
B = A;
row = A;
col = A;

for n=1:Lp,
for l=1:L,
    lim = min(l,n);
    for m=-lim:lim,
        c1 = sqrt((l+m+1)*(l-m+1)/(2*l+1)/(2*l+3))/(l+1);
        c3 = m/(l*(l+1));
        if abs(m) <= l-1
            c2 = sqrt((l+m)*(l-m)/(2*l-1)/(2*l+1))/l;
            c2 = c2*alpz(indAlphaz(l-1,n,m,Lp));
        else
            c2 = 0;
        end
        inds = indAzBz(l,n,m,Lp);
        row(inds) = lm2ind(l,m);
        col(inds) = lm2ind(n,m);
        in1 = indAlphaz(l+1,n,m,Lp);
        in2 = indAlphaz(l,n,m,Lp);
        A(inds) = kr*(c1*alpz(in1) + c2) + alpz(in2);
        B(inds) = 1i*kr*c3*alpz(in2);
    end
end
end
if nargin == 6 && strcmp(normstr,'norm')
    for n=1:Lp,
    for l=1:L,
        lim = min(l,n);
        for m=-lim:lim,
            const = sqrt(l*(l+1))/sqrt(n*(n+1));
```

```
            inds = indAzBz(l,n,m,Lp);
            A(inds) = const*A(inds);
            B(inds) = const*B(inds);
        end
    end
    end
end

function Nl = NAzBz(L,Lp)
% Number of non-zero elements in sparse vector axial translation matrix
%
% L:    Maximum row degree l=1:L
% Lp:   Maximum column degree lp=1:Lp
%
% Nl:   Number of nonzero elements

if L == 1 && Lp == 1
    Nl = 3;
elseif Lp == 1 && L > 1
    Nl = 3*L;
elseif L == 1 && Lp > 1
    Nl = 3*Lp;
elseif L > 1 && Lp > 1 && L > Lp
    Nl = -1/3*Lp^3 + L*Lp^2 + 2*L*Lp + 1/3*Lp;
elseif L > 1 && Lp > 1 && Lp > L
        Nl = -1/3*L^3 + Lp*L^2 + 2*L*Lp + 1/3*L;
elseif L > 1 && Lp > 1 && Lp == L
    Nl = L^2 + 5*L + 2/3*Lp^3 + Lp^2 - 14/3*Lp;
else
    disp('bad index')
    return
end
Nl = round(Nl);

function [ind] = indAzBz(l,lp,m,Lp)
% Row-major linear index of the sparse vector axial translation matrix
%
% l,lp,m:   Harmonix index (l,lp,m)
% Lp:       Maximum column degree lp=1:Lp
%
% ind:      Sparse matrix linear index
%
% Dependencies: NAzBz

if m > min(l,lp) || l < 1 || lp < 1
    disp('bad index')
else
    if l == 1
        ind1 = 0;
    else
        ind1 = NAzBz(l-1,Lp);
    end
    if lp == 1
        ind2 = 0;
    elseif l+1>=lp && ((l==1 && lp>=2) || (lp==2 && l>1))
        ind2 = 3;
    elseif l+1<lp && ((l==1 && lp>=2) || (lp==2 && l>1))
        ind2 = 2*l*(lp-2) + lp + 1;
    elseif l > 1 && (l + 1 >= lp) && l > 2
        ind2 = lp^2 - 1;
    else
        ind2 = -l^2 + l + (2*l + 1)*(lp - 1);
    end
    ind3 = min(l,lp) + m + 1;
    ind = ind1 + ind2 + ind3;
end
ind = round(ind);
```

5.4.3 Diagonalized Vector Translation Matrix

The routine `vectorTranslation` takes as input a vector of expansion coefficients in frame i, $\mathbf{a}$ and $\mathbf{b}$, the maximum degrees L and L' of the translation matrix and the translation vector $\mathbf{r}_{ji}$ in Cartesian coordinates, and returns expansion coefficients in frame j, $\mathbf{c}$ and $\mathbf{d}$. The lengths of $\mathbf{a}$ and $\mathbf{b}$ should be L'^2+2L'. Use `'rg'` for the regular form and `'norm'` for fully normalized vector wave functions. This uses the routines for the sparse versions of the rotation matrix and axial translation matrices to save memory. If the pair of matrices is desired, then include the optional third and fourth outputs.

```
function [clm dlm A B] = vectorTranslation(alm,blm,L,Lp,k,xji,rgstr,normstr)
% Vector translation matrix
%
% alm,blm:      Expansion coefficients in frame i
% L,Lp:         Maximum row/col degrees of the translation matrix
% xji:          3x1 Cartesian vector the points from frame i to frame j
% rgstr:        Use 'rg' for regular form of translation matrix
% normstr:      Use 'norm' for normalized vector wave functions
%
% clm,dlm:      Expansion coefficients in frame j
% A,B:          [optional] Output the pair of full translation matrices
%
% Dependencies: cart2sph,AzBzSparse,DlmpSparseFast,NDlmpSparse

NL = L^2 + 2*L;
NLp = Lp^2 + 2*Lp;
Lmax = max([L Lp]);
Lmin = min([L Lp]);
NLmax = max([NL NLp]);
NLmin = min([NL NLp]);
alm = alm(:);
blm = blm(:);
if length(alm) ~= NLp || length(blm) ~= NLp
    error('alm/blm array lengths do not match Lp')
end

% convert translation vector to spherical
[rji thetaji phiji] = cart2sph(xji(1),xji(2),xji(3));

% compute sparse axial translation matrices
[row col Az Bz] = AzBzSparse(L,Lp,k,rji,rgstr,normstr);
Az = sparse(row,col,Az,NL,NLp);
Bz = sparse(row,col,Bz,NL,NLp);

% compute rotation matrix for larger of L and Lp
[row col D] = DlmpSparseFast(Lmax,phiji+pi/2,thetaji,0);
D1 = sparse(row,col,D,NLmax,NLmax);

% grab the matrix entries for the lesser of L and Lp
Ngrab = NDlmpSparse(Lmin);
D2 = sparse(row(1:Ngrab),col(1:Ngrab),D(1:Ngrab),NLmin,NLmin);

% mutliply depending on L and Lp
if Lp >= L
    tmpalm = D1*alm;
    tmpblm = D1*blm;
    clm = D2'*(Az*tmpalm + Bz*tmpblm);
    dlm = D2'*(Bz*tmpalm + Az*tmpblm);
else
    tmpalm = D2*alm;
    tmpblm = D2*blm;
    clm = D1'*(Az*tmpalm + Bz*tmpblm);
    dlm = D1'*(Bz*tmpalm + Az*tmpblm);
end

if nargout == 4
    if Lp >= L
        A = D2'*Az*D1;
        B = D2'*Bz*D1;
    else
        A = D1'*Az*D2;
        B = D1'*Bz*D2;
    end
end
```

5.4.4 Full Vector Translation Matrix

Like the full scalar translation matrix, recursion relations to compute the full vector translation matrices were derived in [24]. Two versions are available: the first iterates on the vector matrices from scratch, the second iterates on the full scalar translation matrix. We implement the second version, which requires computing the scalar translation matrix at a maximum harmonic degree one higher than that of the vector matrices.

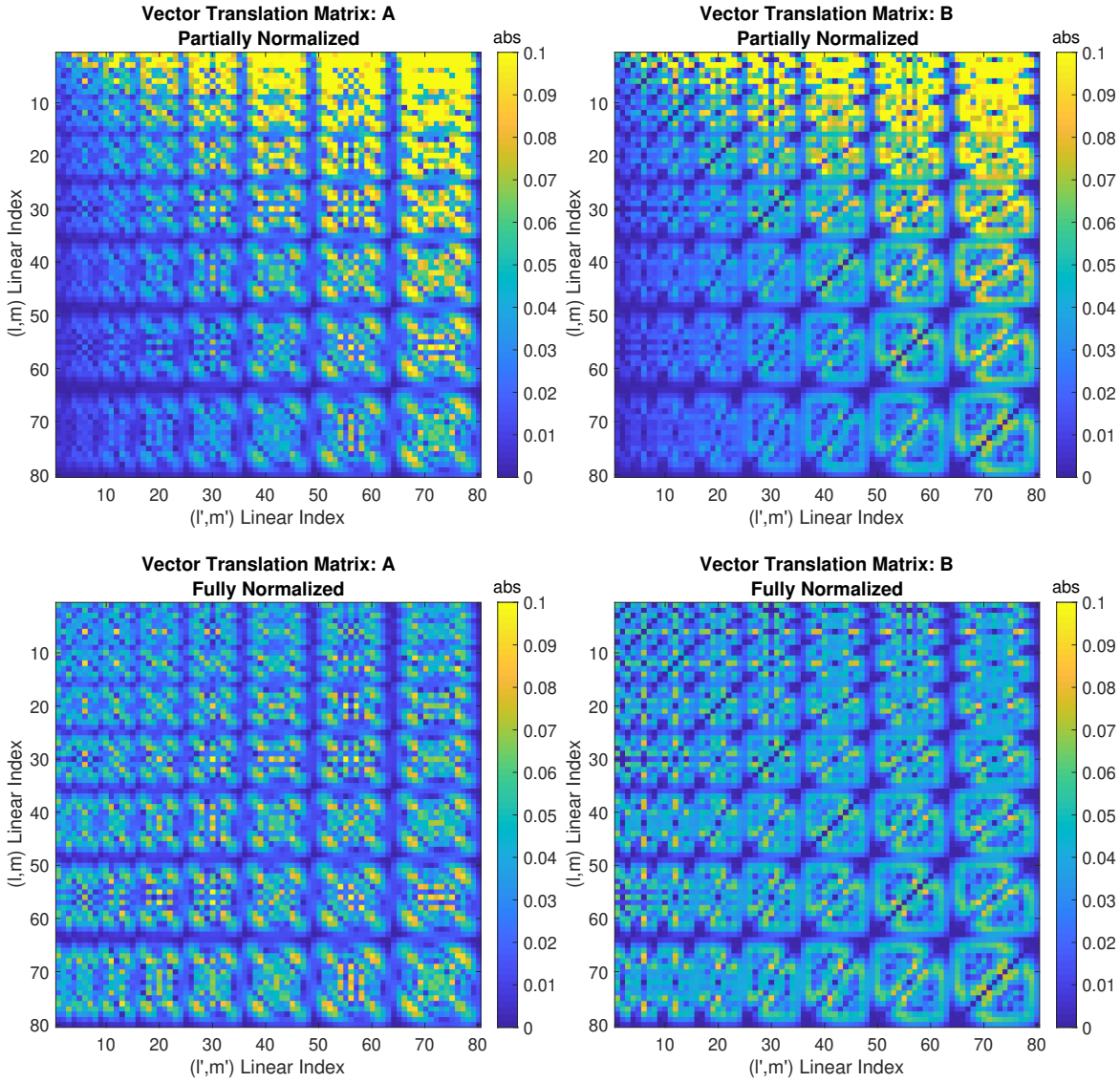


Figure 5.7: Vector translation matrices for $L = L' = 8$, $k = 1$, $r_{ji} = [12, 5, 15]$. Top row: matrices for partially normalized vector wave functions. Bottom row: matrices for fully normalized vector wave functions.

Adopting the notation from [24], which uses the pairings $(\nu\mu, nm)$ for our $(lm, l'm')$, the recurrence relations for the two vector translation matrices derived from the scalar translation matrix are

$$\begin{aligned} A_{\nu\mu,nm} &= \alpha_{\nu\mu,nm} + c_{1,\nu\mu}\alpha_{\nu+1,\mu-1,nm} + c_{2,\nu\mu}\alpha_{\nu-1,\mu-1,nm} + c_{3,\nu\mu}\alpha_{\nu+1,\mu+1,nm} + \\ &\quad c_{4,\nu\mu}\alpha_{\nu-1,\mu+1,nm} + c_{5,\nu\mu}\alpha_{\nu+1,\mu,nm} + c_{6,\nu\mu}\alpha_{\nu-1,\mu,nm} \qquad (5.47) \\ B_{\nu\mu,nm} &= c_{7,\nu\mu}\alpha_{\nu\mu,nm} + c_{8,\nu\mu}\alpha_{\nu,\mu+1,nm} + c_{9,\nu\mu}\alpha_{\nu,\mu-1,nm} \qquad (5.48) \end{aligned}$$

Note, (5.47) and (5.48) only operate on the row indices (ν, μ). The coefficients are

$$c_{1,\nu\mu} = \frac{kr\sin\theta e^{-i\phi}}{2(\nu+1)}\sqrt{\frac{(\nu-\mu+2)(\nu-\mu+1)}{(2\nu+1)(2\nu+3)}} \tag{5.49}$$

$$c_{2,\nu\mu} = -\frac{kr\sin\theta e^{-i\phi}}{2\nu}\sqrt{\frac{(\nu+\mu-1)(\nu+\mu)}{(2\nu-1)(2\nu+1)}} \tag{5.50}$$

$$c_{3,\nu\mu} = -\frac{kr\sin\theta e^{i\phi}}{2(\nu+1)}\sqrt{\frac{(\nu+\mu+2)(\nu+\mu+1)}{(2\nu+1)(2\nu+3)}} \tag{5.51}$$

$$c_{4,\nu\mu} = \frac{kr\sin\theta e^{i\phi}}{2\nu}\sqrt{\frac{(\nu-\mu)(\nu-\mu-1)}{(2\nu-1)(2\nu+1)}} \tag{5.52}$$

$$c_{5,\nu\mu} = \frac{kr\cos\theta}{\nu+1}\sqrt{\frac{(\nu+\mu+1)(\nu-\mu+1)}{(2\nu+1)(2\nu+3)}} \tag{5.53}$$

$$c_{6,\nu\mu} = \frac{kr\cos\theta}{\nu}\sqrt{\frac{(\nu+\mu)(\nu-\mu)}{(2\nu-1)(2\nu+1)}} \tag{5.54}$$

$$c_{7,\nu\mu} = \frac{i\mu kr\cos\theta}{\nu(\nu+1)} \tag{5.55}$$

$$c_{8,\nu\mu} = \frac{ikr\sin\theta e^{i\phi}}{2\nu(\nu+1)}\sqrt{(\nu-\mu)(\nu+\mu+1)} \tag{5.56}$$

$$c_{9,\nu\mu} = \frac{ikr\sin\theta e^{-i\phi}}{2\nu(\nu+1)}\sqrt{(\nu+\mu)(\nu-\mu+1)} \tag{5.57}$$

where (r, θ, ϕ) are the spherical coordinates of the vector, $\mathbf{r}_{ji}$, that points from the originating frame to the translated frame.

The routine `AB` takes as input the maximum harmonic degrees L and L', wavenumber k, and translation vector, r_{ji}, and returns full vector translation matrices A and B. It calls `alpha` to compute the full scalar translation matrix. Use string switch `'rg'` for the regular form of the matrices, and `'norm'` for fully normalized vector wave functions.

```
function [A B] = AB(L,Lp,k,rji,rgstr,normstr)
% Vector translation matrices via the scalar translation matrix
%
% L,Lp:     Maximum row/column harmonic degree
% k:        Background wave number
% rji:      [x,y,z] vector from the source to receiver frame
% rgstr:    'rg' for the regular form: Rg{A} and Rg{B}
% normstr:  'norm' for fully normalized wave functions
%
% A,B:      Translation matrices size N1 x N2
%           N1 = L^2 + 2*L
%           N2 = Lp^2 + 2*Lp
%
% Dependencies: lm2ind, alpha

if nargin == 4
    rgstr = [];
    normstr = [];
end
% convert rji to spherical coordinates
[r th phi] = cart2sph(rji(1),rji(2),rji(3));

% dimensions of the matrix
tot1 = L^2 + 2*L;  % row dimension
tot2 = Lp^2 + 2*Lp;  % col dimension

% preallocation
```

```
A = zeros(tot1,tot2);
B = zeros(tot1,tot2);

% compute alpha (or beta = Rg{alpha}) for L1+1, and L2+1
[alphaLp1] = alpha(L+1,Lp+1,k,rji,rgstr);

% the columns of A and B are shifted one column to the right
% relative to the columns of alpha.
grab = 2:(tot2+1);

% constants used in the interations
const1 = k*r*sin(th)*exp(-i*phi)/2;
const2 = -const1;
const3 = -k*r*sin(th)*exp(i*phi)/2;
const4 = -const3;
const5 = k*r*cos(th);
const6 = const5;
const7 = 1i*const5;
const8 = 1i*k*r*sin(th)*exp(i*phi)/2;
const9 = 1i*k*r*sin(th)*exp(-i*phi)/2;

% loop over rows of the vector matrices
for v=1:L,
for u=-v:v,
    % precomputed linear indecies
    vp = v+1;
    vm = v-1;
    up = u+1;
    um = u-1;
    vu = v^2 + v + u + 1;
    vu1 = vp^2 + vp + um + 1;
    vu2 = vm^2 + vm + um + 1;
    vu3 = vp^2 + vp + up + 1;
    vu4 = vm^2 + vm + up + 1;
    vu5 = vp^2 + vp + u + 1;
    vu6 = vm^2 + vm + u + 1;
    vu8 = v^2 + v + up + 1;
    vu9 = v^2 + v + um + 1;

    % iteration coefficients
    c1 = const1*(1/(v+1))*sqrt((v-u+2)*(v-u+1)/(2*v+1)/(2*v+3));
    c2 = const2*(1/v)*sqrt((v+u-1)*(v+u)/(2*v-1)/(2*v+1));
    c3 = const3*(1/(v+1))*sqrt((v+u+2)*(v+u+1)/(2*v+1)/(2*v+3));
    c4 = const4*(1/v)*sqrt((v-u)*(v-u-1)/(2*v-1)/(2*v+1));
    c5 = const5*(1/(v+1))*sqrt((v+u+1)*(v-u+1)/(2*v+1)/(2*v+3));
    c6 = const6*(1/v)*sqrt((v+u)*(v-u)/(2*v-1)/(2*v+1));
    c7 = const7*u*(1/(v*(v+1)));
    c8 = const8*(1/(v*(v+1)))*sqrt((v-u)*(v+u+1));
    c9 = const9*(1/(v*(v+1)))*sqrt((v+u)*(v-u+1));

    % conditional statements to handle index limits
    if um < -vp || um > vp, tmp1 = 0;
    else tmp1 = c1*alphaLp1(vu1,grab); end
    if um < -vm || um > vm, tmp2 = 0;
    else tmp2 = c2*alphaLp1(vu2,grab); end
    if up < -vp || up > vp, tmp3 = 0;
    else tmp3 = c3*alphaLp1(vu3,grab); end
    if up < -vm || up > vm, tmp4 = 0;
    else tmp4 = c4*alphaLp1(vu4,grab); end
    if up < -vp || up > vp, tmp5 = 0;
    else tmp5 = c5*alphaLp1(vu5,grab); end
    if u < -vm || u > vm, tmp6 = 0;
    else tmp6 = c6*alphaLp1(vu6,grab); end
    if up < -v || up > v, tmp8 = 0;
    else tmp8 = c8*alphaLp1(vu8,grab); end
    if um < -v || um > v, tmp9 = 0;
    else tmp9 = c9*alphaLp1(vu9,grab); end
```

```
    % translation matricies
    A(vu-1,:) = alphaLp1(vu,grab) + tmp1 + tmp2 + tmp3 + tmp4 + tmp5 + tmp6;
    B(vu-1,:) = c7*alphaLp1(vu,grab) + tmp8 + tmp9;
end
end

% apply normalization if needed
if nargin == 6 && strcmp(normstr,'norm')
    for n=1:Lp,
    for mp=-n:n,
    for l=1:L,
    for m=-l:l,
        const = sqrt(l*(l+1))/sqrt(n*(n+1));
        A(lm2ind(l,m),lm2ind(n,mp)) = const*A(lm2ind(l,m),lm2ind(n,mp));
        B(lm2ind(l,m),lm2ind(n,mp)) = const*B(lm2ind(l,m),lm2ind(n,mp));
    end
    end
    end
    end
end
```

Chapter 6

S-Matrix

The scattering matrix, or S-matrix (also called the scattering function matrix, [10]), embeds the scattering behavior of an object as a mapping between incident and scattered plane-waves. It transforms between the incident/scattered directions as well as incident/scattered polarizations. This chapter gives the basic definition of the S-matrix, its relation to radar cross sections, routines for the S-matrix of simple objects, and the derivation of an object S-matrix under the Born approximation.

6.1 Definition

From [10], let an incident wave be a plane wave with wave vector $\mathbf{k}_i$, the polarization of which is decomposed into two linearly independent vectors that are perpendicular to $\hat{k}_i$, as

$$\mathbf{E}_i = (E_{pi}\hat{p}_i + E_{qi}\hat{q}_i)\, e^{i\mathbf{k}_i\cdot\mathbf{r}} \tag{6.1}$$

where $\mathbf{r}$ is the position vector, $\mathbf{k}_i = k\hat{k}_i$ and $\hat{p}_i$, $\hat{q}_i$ and $\hat{k}_i$ form an orthonormal system. The scattered field is then defined

$$\mathbf{E}_s = (E_{ps}\hat{p}_s + E_{qs}\hat{q}_s)\,\frac{e^{ikr}}{r} \tag{6.2}$$

where the scattered field polarization vectors $\hat{p}_s$ and $\hat{q}_s$ form a right-handed orthonormal system with the scattered field direction $\hat{k}_s$. The scattered field components are linearly related to the incident components through the scattering matrix, or S-matrix,

$$\begin{bmatrix} E_{ps}(\hat{k}_s) \\ E_{qs}(\hat{k}_s) \end{bmatrix} = \begin{bmatrix} S_{pp}(\hat{k}_s,\hat{k}_i) & S_{pq}(\hat{k}_s,\hat{k}_i) \\ S_{qp}(\hat{k}_s,\hat{k}_i) & S_{qq}(\hat{k}_s,\hat{k}_i) \end{bmatrix} \begin{bmatrix} E_{pi}(\hat{k}_i) \\ E_{qi}(\hat{k}_i) \end{bmatrix} \tag{6.3}$$

These definitions for the fields and S-matrix give them the following units: the incident field amplitudes E_{pi} and E_{qi} have units of electric field (V/m), the S-matrix elements have units of length (m), and the scattering field amplitudes E_{ps} and E_{qs} have units of (Vm/m = V) because the length dimension of the factor of $1/r$ needs to be included. This depends on convention, because some definitions of the S-matrix use $1/kr$ outside the S-matrix, which makes the S-matrix unitless.

We use the wave vector and polarization convention of [10]. The polarizations are taken as $\hat{p} = \hat{v}$ and $\hat{q} = \hat{h}$ with wave vector directions

$$\hat{k}_i = \sin\theta_i\cos\phi_i\hat{x} + \sin\theta_i\sin\phi_i\hat{y} + \cos\theta_i\hat{z} \tag{6.4}$$
$$\hat{v}_i = \cos\theta_i\cos\phi_i\hat{x} + \cos\theta_i\sin\phi_i\hat{y} - \sin\theta_i\hat{z} \tag{6.5}$$
$$\hat{h}_i = -\sin\phi_i\hat{x} + \cos\phi_i\hat{y} \tag{6.6}$$
$$\hat{k}_s = \sin\theta_s\cos\phi_s\hat{x} + \sin\theta_s\sin\phi_s\hat{y} + \cos\theta_s\hat{z} \tag{6.7}$$
$$\hat{v}_s = \cos\theta_s\cos\phi_s\hat{x} + \cos\theta_s\sin\phi_s\hat{y} - \sin\theta_s\hat{z} \tag{6.8}$$
$$\hat{h}_s = -\sin\phi_s\hat{x} + \cos\phi_s\hat{y} \tag{6.9}$$

where θ, ϕ are spherical directions of the propagation waves. Here, the wave vectors are treated as radial vectors, and $\hat{v}$ and $\hat{h}$ are equivalent to the $\hat{\theta}$ and $\hat{\phi}$ spherical unit vectors. The polarizations for $\hat{z}$ propagating waves ($\theta = 0$ or $\theta = \pi$) are well-defined by using $\phi = 0$ as a reference

$$\begin{aligned}
\hat{k}(0,0) &= \hat{z} && (6.10)\\
\hat{v}(0,0) &= \hat{x} && (6.11)\\
\hat{h}(0,0) &= \hat{y} && (6.12)\\
\hat{k}(\pi,0) &= -\hat{z} && (6.13)\\
\hat{v}(\pi,0) &= -\hat{x} && (6.14)\\
\hat{h}(\pi,0) &= -\hat{y} && (6.15)
\end{aligned}$$

These special cases can be used as definitions of the $\hat{\theta}$ and $\hat{\phi}$ unit vectors at the poles.

6.2 Radar Cross Sections from S-matrix

We list different radar cross sections and their relations to the S-matrix. Most of these can be found in [10].

Radar Cross Section The bistatic radar cross section is defined

$$\begin{aligned}
\sigma_{pq}(\hat{k}_s, \hat{k}_i) &= \lim_{r\to\infty} 4\pi r^2 \frac{|\hat{p}\cdot\mathbf{E}_s|^2}{|\hat{q}\cdot\mathbf{E}_i|^2} && (6.16)\\
&= 4\pi|S_{pq}|^2 && (6.17)
\end{aligned}$$

where p and q are any two orthogonal polarizations. The units of radar cross section are area or length-squared, showing again that S_{pq} has units of length.

Scattering Cross Section The total scattering cross section for incident direction $\hat{k}_i$ and arbitrary incident polarization $\hat{\beta}$, such that $\hat{\beta}\cdot\hat{k}_i = 0$, is computed as the integral over scattered field directions $\hat{k}_s$ as

$$\sigma_{s\beta}(\hat{k}_i) = \int \left(|S_{p\beta}(\hat{k}_s,\hat{k}_i)|^2 + |S_{q\beta}(\hat{k}_s,\hat{k}_i)|^2\right) d\Omega_s \tag{6.18}$$

where

$$\begin{bmatrix} S_{p\beta} \\ S_{q\beta} \end{bmatrix} = \begin{bmatrix} S_{pp} & S_{pq} \\ S_{qp} & S_{qq} \end{bmatrix} \begin{bmatrix} \hat{p}\cdot\hat{\beta} \\ \hat{q}\cdot\hat{\beta} \end{bmatrix} \tag{6.19}$$

and β is the angle of the incident polarization relative to the two orthogonal polarizations of the S-matrix, which is

$$\hat{\beta} = \cos\beta\hat{p} + \sin\beta\hat{q} \tag{6.20}$$

Extinction Cross Section The extinction cross section (or total cross section) for a given incident direction is given by the optical theorem, [26], as the imaginary part of the co-polarized scattered field in the forward direction

$$\sigma_{ext,\beta}(\hat{k}_i) = \frac{4\pi}{k}\mathrm{Im}\left[S_{\beta\beta}(\hat{k}_i,\hat{k}_i)\right] \tag{6.21}$$

Absorption Cross Section The absorption cross section is defined as the power absorbed by the scattering objected divided by the power scattered. It can be written in terms of the extinction and total cross sections as

$$\sigma_a = \sigma_{ext} - \sigma_s \tag{6.22}$$

This subtraction can sometimes be numerically inaccurate, and so the absorption cross section can also be defined directly in terms of the internal field, [27].

Polarization and Orientation Averaged Scattering Cross Section The polarization and orientation averaged scattered cross section is given by the integral of the scattering cross section over all possible incident directions and polarizations, including normalization:

$$\langle \sigma \rangle = \frac{1}{2\pi} \int_0^{2\pi} \left(\frac{1}{4\pi} \int \sigma_{s\beta}(\hat{k}_i) d\Omega_i \right) d\beta \tag{6.23}$$

This requires projecting a given incident polarization β onto $\hat{p}$ and $\hat{q}$, computing the projected S-matrix elements (6.19), computing the scattering cross section for all incident directions, then integrating over incident direction and incident angle β. Computing this does not require one to explicitly rotate or recompute the S-matrix once it is known.

Scattering Efficiencies The quantities above can be reinterpreted as efficiencies by dividing by the cross sectional area of the target. For example, the scattering efficiency for a given incident direction is given by the scattering cross section divided by cross sectional area of the target, A,

$$Q_{sca} = \frac{\sigma_s}{A} \tag{6.24}$$

The same can be applied to extinction and absorption cross sections.

6.3 S-matrix Rotation

An S-matrix will often be computed in a reference frame aligned naturally with the geometry of the object. If an object is rotated relative to a global frame, then the incident and scattered directions and polarization in the global frame will appear to rotate in the frame of the object. It can be efficient to transform the global wave vectors and polarizations such that the S-matrix is evaluated in the frame of the object, rather than try to obtain the S-matrix of the rotated object in the global frame. Both [10] and [28] give equations and procedures for rotating an object with cylindrical symmetry. We give a general procedure next.

Let an object rotation be described by ZXZ Euler angles (α, β, γ) with corresponding rotation matrix $\mathbf{R}$. $\mathbf{R}$ describes a forward rotation, i.e., a point rides with the rotated frame. The inverse rotation is $\mathbf{R}^t$, where points are fixed in the global frame and we view them as though we ride with the rotating frame. The inverse rotation is first applied to the incident and scattered directions of the global frame so that they are viewed from the rotated object frame, let these be $(\theta_{os}, \phi_{os}; \theta_{oi}, \phi_{oi})$. The polarization vectors are then evaluated in the object frame at these points giving $(\hat{v}_{os}, \hat{h}_{os}; \hat{v}_{oi}, \hat{h}_{oi})$. The object-frame polarization vectors now need to be viewed from the global frame. This is done by rotating the object-frame polarization vectors with an forward rotation, so that they appear as vectors in the global frame:

$$\hat{v}'_o = \mathbf{R}(\hat{v}_o) \tag{6.25}$$

In matrix notation, S-matrix of the rotated object, as viewed in the global frame, is given by

$$\begin{bmatrix} S_{vv}(\theta_s, \phi_s; \theta_i, \phi_i) & S_{vh}(\theta_s, \phi_s; \theta_i, \phi_i) \\ S_{hv}(\theta_s, \phi_s; \theta_i, \phi_i) & S_{hh}(\theta_s, \phi_s; \theta_i, \phi_i) \end{bmatrix} = \begin{bmatrix} \hat{v}_s \cdot \hat{v}'_{os} & \hat{v}_s \cdot \hat{h}'_{os} \\ \hat{h}_s \cdot \hat{v}'_{os} & \hat{h}_s \cdot \hat{h}'_{os} \end{bmatrix} \cdot \begin{bmatrix} S_{vv}(\theta_{os}, \phi_{os}; \theta_{oi}, \phi_{oi}) & S_{vh}(\theta_{os}, \phi_{os}; \theta_{oi}, \phi_{oi}) \\ S_{hv}(\theta_{os}, \phi_{os}; \theta_{oi}, \phi_{oi}) & S_{hh}(\theta_{os}, \phi_{os}; \theta_{oi}, \phi_{oi}) \end{bmatrix} \cdot \begin{bmatrix} \hat{v}'_{oi} \cdot \hat{v}_i & \hat{v}'_{oi} \cdot \hat{h}_i \\ \hat{h}'_{oi} \cdot \hat{v}_i & \hat{h}'_{oi} \cdot \hat{h}_i \end{bmatrix} \tag{6.26}$$

The inner matrix is the object S-matrix evaluated in its native frame using points rotated from the global frame. The outer two matrices decompose and project the polarization vectors between the two frames.

6.4 Object S-matrix

6.4.1 Thin Circular Cylinder

From [29, 30] the far field scattering solution for a thin dielectric cylinder with cross sectional area much smaller than a wavelength and oriented along the z axis is given by

$$\mathbf{E}_s = -E_o \frac{e^{ikr}}{4\pi r} k^2 LA \left[\hat{k}_s \times \hat{k}_s \times (\mathbf{P} \cdot \hat{e}_i)\right] \mathrm{sinc}(U) \tag{6.27}$$

$$U = \frac{kL}{2}(\cos\theta_s - \cos\theta_i) \tag{6.28}$$

where L is the length, A is the cross sectional area, $\hat{e}_i$ is the incident polarization, k is the background wavenumber, θ_i and θ_s are the incident and scattered angles from the z axis, and $\mathbf{P}$ is the polarization tensor. Also, $\mathrm{sinc}(x) = \sin(x)/x$. The polarization tensor for a homogenous dielectric cylinder with circular cross section is diagonal and given by

$$P_{xx} = 2\frac{\epsilon_r - 1}{\epsilon_r + 1} \tag{6.29}$$

$$P_{yy} = 2\frac{\epsilon_r - 1}{\epsilon_r + 1} \tag{6.30}$$

$$P_{zz} = \epsilon_r - 1 \tag{6.31}$$

where ϵ_r is the relative permittivity. For a cylinder with radius a, the $\hat{v}$ and $\hat{h}$ S-matrix elements are

$$\begin{bmatrix} S_{vv} & S_{vh} \\ S_{hv} & S_{hh} \end{bmatrix} = \frac{k^2 La^2}{4} \mathrm{sinc}(U)\mathbf{C} \tag{6.32}$$

$$\mathbf{C} = \begin{bmatrix} (\hat{v}_s) \cdot (\hat{k}_s \times \hat{k}_s \times (\mathbf{P} \cdot \hat{v}_i)) & (\hat{v}_s) \cdot \left(\hat{k}_s \times \hat{k}_s \times \left(\mathbf{P} \cdot \hat{h}_i\right)\right) \\ (\hat{h}_s) \cdot (\hat{k}_s \times \hat{k}_s \times (\mathbf{P} \cdot \hat{v}_i)) & (\hat{h}_s) \cdot \left(\hat{k}_s \times \hat{k}_s \times \left(\mathbf{P} \cdot \hat{h}_i\right)\right) \end{bmatrix} \tag{6.33}$$

Using (6.4)-(6.9) and simplifying the elements of $\mathbf{C}$ are

$$C_{vv} = P_{zz}\sin\theta_s \sin\theta_i + P_{xx}\cos\theta_s \cos\phi_s \cos\theta_i \cos\phi_i + P_{yy}\cos\theta_s \cos\theta_i \sin\phi_s \sin\phi_i \tag{6.34}$$

$$C_{vh} = \cos\theta_s(P_{yy}\sin\phi_s \cos\phi_i - P_{xx}\cos\phi_s \sin\phi_i) \tag{6.35}$$

$$C_{hv} = \cos\theta_i(P_{yy}\cos\phi_s \sin\phi_i - P_{xx}\sin\phi_s \cos\phi_i) \tag{6.36}$$

$$C_{hh} = P_{yy}\cos\phi_s \cos\phi_i + P_{xx}\sin\phi_s \sin\phi_i \tag{6.37}$$

Applying symmetry $P_{xx} = P_{yy}$ and taking $\phi_i = 0$, these can be simplified to just

$$C_{vv} = P_{zz}\sin\theta_s \sin\theta_i + P_{xx}\cos\theta_s \cos\phi_s \cos\theta_i \tag{6.38}$$

$$C_{vh} = P_{xx}\cos\theta_s \sin\phi_s \tag{6.39}$$

$$C_{hv} = -P_{xx}\cos\theta_i \sin\phi_s \tag{6.40}$$

$$C_{hh} = P_{xx}\cos\phi_s \tag{6.41}$$

This solution is valid for edge-on incidence, and the hh and vv fields become equal, which we expect. This solution is valid for $|n|ka \ll 1$ and $a \ll L$, where $n = \sqrt{\epsilon_r}$, [31].

The routine `smatrix_thin_circular_cylinder` returns the S-matrix of a thin circular cylinder. It takes the parameters of a single cylinder and computes the four block S-matrices for any number of incident and scattered directions. The block matrices have scattered directions in rows and incident directions in columns. The polarizations $\hat{v}$ and $\hat{h}$ are treated as spherical $\hat{\theta}$ and $\hat{\phi}$ in the forward scattering convention.

```
function [Svv Svh Shv Shh] = smatrix_thin_circular_cylinder(k,a,L,er,theta_s,phi_s,theta_i,phi_i)
% S-matrix of a thin circular cylinder, oriented along the z-axis
%
% k:                  Background wavenumber (1/m)
% a:                  cylinder radius (m)
% L:                  cylinder length (m)
% er:                 cylinder complex relative permittivity
% theta_s,phi_s:      Scatttered directions, radians, any size, sz_s = size(theta_s)
% theta_i,phi_i:      Incident directions, radians, any size, sz_i = size(theta_i)
%
% Svv,Svh,Shv,Shh:    S-matrix block matrices h/v components,
%                     Size [sz_s sz_i], concatenated sizes of the input
%                     incident and scattered directions

% size of incident/scattered direction arrays
sz_i = size(theta_i);
sz_s = size(theta_s);

% total number of incident/scattered directions
Ni = numel(theta_i);
Ns = numel(theta_s);

% reshape the inputs
theta_i = theta_i(:);
phi_i = phi_i(:);
theta_s = theta_s(:);
phi_s = phi_s(:);

% ndgrid reshaped inputs
[Th_s Th_i] = ndgrid(theta_s,theta_i);
[Phi_s Phi_i] = ndgrid(phi_s,phi_i);

% polarization vector
Pxx = 2*(er - 1)./(er + 1);
Pyy = Pxx;
Pzz = er - 1;

% C matrix
sth_s = sin(Th_s);
cth_s = cos(Th_s);
sph_s = sin(Phi_s);
cph_s = cos(Phi_s);
sth_i = sin(Th_i);
cth_i = cos(Th_i);
sph_i = sin(Phi_i);
cph_i = cos(Phi_i);
Cvv = Pzz.*sth_s.*sth_i + Pxx.*cth_s.*cph_s.*cth_i.*cph_i + Pyy.*cth_s.*cth_i.*sph_s.*sph_i;
Cvh = cth_s.*(Pyy.*sph_s.*cph_i - Pxx.*cph_s.*sph_i);
Chv = cth_i.*(Pyy.*cph_s.*sph_i - Pxx.*sph_s.*cph_i);
Chh = Pyy.*cph_s.*cph_i + Pxx.*sph_s.*sph_i;

% constant
const = k^2*L*a^2/4;

% sinc function and constant
U = k*L/2*(cth_s - cth_i);
sincU = const*sinc(U/pi);  % matlab's sinc is sin(pi*x)/(pi*x);

% Smatrix elements and reshape
Svv = squeeze(reshape(sincU.*Cvv,[sz_s sz_i]));
Svh = squeeze(reshape(sincU.*Cvh,[sz_s sz_i]));
Shv = squeeze(reshape(sincU.*Chv,[sz_s sz_i]));
Shh = squeeze(reshape(sincU.*Chh,[sz_s sz_i]));
```

6.4.2 Thin Circular Disk

In [31] the scattering for both needles and disks is derived. The results are similar to pervious section, except for different coordinate systems. The far field scattered field for a thin disk is

$$\begin{aligned}
\mathbf{E}_s &= -E_o \frac{e^{ikr}}{2\pi r} k^2 LA \left[\hat{k}_s \times \hat{k}_s \times (\mathbf{P} \cdot \hat{e}_i)\right] \text{jinc}(U) && (6.42)\\
U &= kL\Omega && (6.43)\\
\Omega^2 &= (\sin\theta_i \cos\phi_i - \sin\theta_s \cos\phi_s)^2 + (\sin\theta_i \sin\phi_i - \sin\theta_s \sin\phi_s)^2 && (6.44)
\end{aligned}$$

where L is the disk thickness, A is cross sectional area of the disk, and $\text{jinc}(x) = J_1(x)/x$ where $J_1(x)$ is the Bessel function of degree 1. The polarization tensor of the circular disk is

$$\begin{aligned}
P_{xx} &= \epsilon_r - 1 && (6.45)\\
P_{yy} &= \epsilon_r - 1 && (6.46)\\
P_{zz} &= \frac{\epsilon_r - 1}{\epsilon_r} && (6.47)
\end{aligned}$$

Then the S-matrix elements are

$$\begin{bmatrix} S_{vv} & S_{vh} \\ S_{hv} & S_{hh} \end{bmatrix} = \frac{k^2 La^2}{2} \frac{J_1(U)}{U} \mathbf{C} \qquad (6.48)$$

where $\mathbf{C}$ is given by (6.33). In comparing [31] to [29, 30], the factor of 2 appears correct between the cylinder and disk scattered field. This solution is valid for $|n|kL \ll 1$ and $L \ll a$, where $n = \sqrt{\epsilon_r}$, [31].

The routine `smatrix_thin_circular_disk` returns the S-matrix of a thin circular disk. It takes the parameters of a single disk and computes the four block S-matrices for any number of incident and scattered directions. The block matrices have scattered directions in rows and incident directions in columns. The polarizations $\hat{v}$ and $\hat{h}$ are treated as spherical $\hat{\theta}$ and $\hat{\phi}$ in the forward scattering convention.

```
function [Svv Svh Shv Shh] = smatrix_thin_circular_disk(k,a,L,er,theta_s,phi_s,theta_i,phi_i)
% S-matrix of a thin circular disk, oriented with normal along the z-axis
%
% k:                  Background wavenumber (1/m)
% a:                  disk radius (m)
% L:                  disk thickness (m)
% er:                 disk complex relative permittivity
% theta_s,phi_s:      Scatttered directions, radians, any size, sz_s = size(theta_s)
% theta_i,phi_i:      Incident directions, radians, any size, sz_i = size(theta_i)
%
% Svv,Svh,Shv,Shh:    S-matrix block matrices h/v components,
%                     Size [sz_s sz_i], concatenated sizes of the input
%                     incident and scattered directions
%
% Dependencies: jinc

% size of incident/scattered direction arrays
sz_i = size(theta_i);
sz_s = size(theta_s);

% total number of incident/scattered directions
Ni = numel(theta_i);
Ns = numel(theta_s);

% reshape the inputs
theta_i = theta_i(:);
phi_i = phi_i(:);
theta_s = theta_s(:);
phi_s = phi_s(:);

% ndgrid reshaped inputs
[Th_s Th_i] = ndgrid(theta_s,theta_i);
```

```
[Phi_s Phi_i] = ndgrid(phi_s,phi_i);

% polarization vector
Pxx = er - 1;
Pyy = Pxx;
Pzz = (er - 1)./er;

% C matrix
sth_s = sin(Th_s);
cth_s = cos(Th_s);
sph_s = sin(Phi_s);
cph_s = cos(Phi_s);
sth_i = sin(Th_i);
cth_i = cos(Th_i);
sph_i = sin(Phi_i);
cph_i = cos(Phi_i);
Cvv = Pzz.*sth_s.*sth_i + Pxx.*cth_s.*cph_s.*cth_i.*cph_i + Pyy.*cth_s.*cth_i.*sph_s.*sph_i;
Cvh = cth_s.*(Pyy.*sph_s.*cph_i - Pxx.*cph_s.*sph_i);
Chv = cth_i.*(Pyy.*cph_s.*sph_i - Pxx.*sph_s.*cph_i);
Chh = Pyy.*cph_s.*cph_i + Pxx.*sph_s.*sph_i;

% constant
const = k^2*L*a^2/2;

% sinc function and constant
Omega2 = (sth_i.*cph_i - sth_s.*cph_s).^2 + (sth_i.*sph_i - sth_s.*sph_s).^2;
U = k*L*sqrt(Omega);
jincU = const*jinc(U);  % matlab's sinc is sin(pi*x)/(pi*x);

% Smatrix elements and reshape
Svv = squeeze(reshape(jincU.*Cvv,[sz_s sz_i]));
Svh = squeeze(reshape(jincU.*Cvh,[sz_s sz_i]));
Shv = squeeze(reshape(jincU.*Chv,[sz_s sz_i]));
Shh = squeeze(reshape(jincU.*Chh,[sz_s sz_i]));
```

6.5 S-matrix Under the Born Approximation

The far-field Born approximation is explained in Section 2.4. Because the approximation is written in terms of plane waves, it can be cast as an S-matrix. Recalling (2.45), using $\overline{\mathbf{I}} - \hat{r}\hat{r} = \hat{\theta}_s\hat{\theta}_s + \hat{\phi}_s\hat{\phi}_s$, and equating the spherical unit vectors with $\hat{v}$ and $\hat{h}$, we obtain the S-matrix

$$\begin{bmatrix} S_{vv} & S_{vh} \\ S_{hv} & S_{hh} \end{bmatrix} = \frac{1}{4\pi} \begin{bmatrix} \hat{v}_s \cdot \hat{v}_i & \hat{v}_s \cdot \hat{h}_i \\ \hat{h}_s \cdot \hat{v}_i & \hat{h}_s \cdot \hat{h}_i \end{bmatrix} \int O(\mathbf{r}) e^{i(\mathbf{k}_i - \mathbf{k}_s)\cdot\mathbf{r}} dV \tag{6.49}$$

where the object function is given by (2.40) and the wave vectors and polarizations are given by (6.4)-(6.9). This shows again that under the Born approximation, the object does not influence the polarization. The depolarization is merely the projection between incident and scattered polarizations.

When the object is homogenous (6.49) becomes

$$\begin{bmatrix} S_{vv} & S_{vh} \\ S_{hv} & S_{hh} \end{bmatrix} = \frac{1}{4\pi} \begin{bmatrix} \hat{v}_s \cdot \hat{v}_i & \hat{v}_s \cdot \hat{h}_i \\ \hat{h}_s \cdot \hat{v}_i & \hat{h}_s \cdot \hat{h}_i \end{bmatrix} k^2(\epsilon_r - 1) I(\theta_s, \phi_s; \theta_i, \phi_i) \tag{6.50}$$

$$I(\theta_s, \phi_s; \theta_i, \phi_i) = \int e^{i(\mathbf{k}_i - \mathbf{k}_s)\cdot\mathbf{r}} dV \tag{6.51}$$

where k is the wavenumber of the background medium which is also used for the wave vectors, ϵ_r is complex relative permittivity of the object, and I is the volume phase integral. The volume phase integral is the 3D Fourier transform over the domain of the object.

6.6 Volume Phase Integral

The volume phase integral occurs in the S-matrix formulation of the Born approximation for homogeneous dielectric objects, Section 6.5. The volume phase integral for several simple objects is derived below and summarized in Table 6.1. These are just the 3D Fourier transform over the object domain. In general, the volume phase integral can be expressed as a product of the volume of the object times a function that has maximum value of one and that depends on the incident/scattered directions. To facilitate the derivations, (6.51) is written in terms of the wave vector difference, $\mathbf{K}$, as

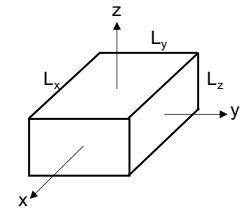
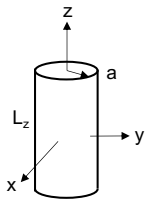
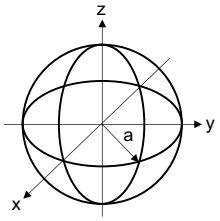
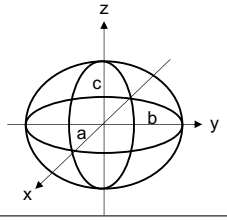
$$I(\theta_s, \phi_s; \theta_i, \phi_i) = \int e^{i\mathbf{K}\cdot\mathbf{r}} dV \tag{6.52}$$

where

$$\begin{aligned}
\mathbf{K} &= \mathbf{k}_i - \mathbf{k}_s = k(\hat{k}_i - \hat{k}_s) && (6.53)\\
\hat{k}_i &= \sin\theta_i \cos\phi_i \hat{x} + \sin\theta_i \sin\phi_i \hat{y} + \cos\theta_i \hat{z} && (6.54)\\
\hat{k}_r &= \sin\theta_r \cos\phi_r \hat{x} + \sin\theta_r \sin\phi_r \hat{y} + \cos\theta_r \hat{z} && (6.55)\\
\mathbf{r} &= x\hat{x} + y\hat{y} + z\hat{z} && (6.56)
\end{aligned}$$

Table 6.1: Table of Volume Phase Integrals

$$I(\theta_s, \phi_s; \theta_i, \phi_i) = \int e^{i\mathbf{K}\cdot\mathbf{r}} dV = V f(\cdot)$$
$$\mathbf{K} = \mathbf{k}_i - \mathbf{k}_s$$

Object	Geometry	V	$f(\cdot)$	Notes
Cuboid		$L_x L_y L_z$	$\text{sinc}\left(\frac{L_x K_x}{2}\right)\text{sinc}\left(\frac{L_y K_y}{2}\right)\text{sinc}\left(\frac{L_z K_z}{2}\right)$	$\text{sinc}(x) = \frac{\sin(x)}{x}$
Circular Cylinder		$\pi a^2 L_z$	$2\frac{J_1(K_\rho a)}{K_\rho a}\text{sinc}\left(\frac{L_z K_z}{2}\right)$	$K_\rho = \sqrt{K_x^2 + K_y^2}$
Sphere		$\frac{4}{3}\pi a^3$	$\frac{3(\sin(Ka) - Ka\cos(Ka))}{(Ka)^3}$	$K = \lvert\mathbf{K}\rvert$
Ellipsoid		$\frac{4}{3}\pi abc$	$\frac{3(\sin(K') - K'\cos(K'))}{K'^3}$	$K' = \lvert\mathbf{K}'\rvert$ $(K'_x, K'_y, K'_z) =$ (aK_x, bK_y, cK_z)

6.6.1 Cuboid

Let a cuboid volume be centered at the origin with side lengths L_x, L_y, L_z and let it be aligned with the Cartesian axes. In Cartesian coordinates, the volume phase integral, (6.52), becomes

$$I = \int_{-L_x/2}^{L_x/2}\int_{-L_y/2}^{L_y/2}\int_{-L_z/2}^{L_z/2} e^{i(K_x x+K_y y+K_z z)}dxdydz \tag{6.57}$$

The integral separates, then using the fact that

$$\int_{-a/2}^{a/2} e^{ibz}dz = \frac{2}{b}\sin\left(\frac{ab}{2}\right) \tag{6.58}$$

and multiplying top and bottom by $L_xL_yL_z$ to convert the sines to sinc functions the volume phase intergral over a cuboid is

$$I = L_xL_yL_z\text{sinc}\left(\frac{L_xK_x}{2}\right)\text{sinc}\left(\frac{L_yK_y}{2}\right)\text{sinc}\left(\frac{L_zK_z}{2}\right) \tag{6.59}$$

where $\text{sinc} = \sin(x)/x$. This is equal to the volume of the cuboid times a product of directionally dependent sinc functions.

6.6.2 Circular Cylinder

Let a circular cylinder be centered at the origin and aligned with the z axis with radius a and length L_z. The volume phase integral, (6.52), in cylindrical coordinates is written

$$I = \int_{-L_z/2}^{L_z/2}\int_0^{2\pi}\int_0^a e^{i(K_x\rho\cos\phi+K_y\rho\sin\phi+K_z z)}\rho d\rho d\phi dz \tag{6.60}$$

where we have used the position vector $\mathbf{r} = \rho\cos\phi\hat{x} + \rho\sin\phi\hat{y} + \hat{z}$. The integral separates as

$$I = \int_0^{2\pi}\int_0^a e^{i(K_x\rho\cos\phi+K_y\rho\sin\phi)}\rho d\rho d\phi\int_{-L_z/2}^{L_z/2} e^{iK_z z}dz \tag{6.61}$$

The last integral is a sinc function in z given by (6.58). Using the identity

$$\int_0^{2\pi} e^{u\cos t+v\sin t}dt = 2\pi I_0\left(\sqrt{u^2+v^2}\right) \tag{6.62}$$

the integral over ϕ evaluates to

$$\int_0^{2\pi} e^{iK_x\rho\cos\phi+K_y\rho\sin\phi}d\phi = 2\pi I_0\left(iK_\rho\rho\right) \tag{6.63}$$

where $K_\rho = \sqrt{K_x^2+K_y^2}$. The quantity K_ρ is the component of the wave vector difference in the X-Y plane. Last, the integral in ρ is evaluated using

$$\int_0^a I_0(icx)xdx = \frac{aJ_1(ac)}{c} \tag{6.64}$$

Putting these together, the volume phase integral over a circular cylinder is

$$I = 2\pi\frac{aJ_1(K_\rho a)}{K_\rho}L_z\text{sinc}\left(\frac{L_zK_z}{2}\right) \tag{6.65}$$

Using the volume of the cylinder, $V = \pi a^2L_z$, this can be written

$$I = V2\frac{J_1(K_\rho a)}{K_\rho a}\text{sinc}\left(\frac{L_zK_z}{2}\right) \tag{6.66}$$

Similar to the other shapes, this is equal to the volume of the cylinder multiplied by a directionally dependent term that has maximum value of 1.

6.6.3 Sphere

Let a spherical volume with radius a be centered at the origin. The volume phase integral, (6.52), can be written in spherical coordinates as

$$I = \int_0^{2\pi} \int_0^{\pi} \int_0^{a} e^{i\mathbf{K}\cdot\mathbf{r}} r^2 \sin\theta dr d\theta d\phi \tag{6.67}$$

From symmetry, the dot product is evaluated relative to a common fixed axis such that $\mathbf{K}\cdot\mathbf{r} = Kr\cos t$ where t is the angle between $\mathbf{K}$ and $\mathbf{r}$. Using this, the integral becomes

$$I = \int_0^{2\pi} \int_0^{\pi} \int_0^{a} e^{iKr\cos t} r^2 \sin t dr dt d\phi \tag{6.68}$$

Next we use the identity

$$j_n(z) = \frac{(-i)^n}{2} \int_0^{\pi} e^{iz\cos u} P_n(\cos u)\sin u du \tag{6.69}$$

where $j_n(z)$ is the spherical Bessel function and $P_n(\cos u)$ is the Legendre polynomial. Applying this and integrating in ϕ (6.68) becomes

$$I = 4\pi \int_0^{a} j_0(Kr) r^2 dr \tag{6.70}$$

Note that $j_0(x) = \sin(x)/x$. This integral is given generally as

$$\int_0^{a} j_0(cx) x^2 dx = \frac{\sin(ac) - ac\cos(ac)}{c^3} \tag{6.71}$$

The volume phase integral over a sphere is given by

$$I = 4\pi \frac{\sin(Ka) - Ka\cos(Ka)}{K^3} \tag{6.72}$$

where K is the magnitude of (6.53). Using the volume of the sphere, $V = 4/3\pi a^3$, this can be written

$$I = V \frac{3(\sin(Ka) - Ka\cos(Ka))}{(Ka)^3} \tag{6.73}$$

The multiplying function acts like a directionally dependent weighting function that has a maximum value of 1 when $Ka \to 0$.

6.6.4 Ellipsoid

Let an ellipsoid be centered on the origin with axes (a, b, c) along XYZ, respectively. We define a new position vector, $\mathbf{r}'$, in spherical coordinates using the change of variables

$$\mathbf{r}' = ra\sin\theta\cos\phi\hat{x} + rb\sin\theta\sin\phi\hat{y} + rc\cos\theta\hat{z} \tag{6.74}$$

$$dV = abcr^2\sin\theta dr d\theta d\phi \tag{6.75}$$

The volume phase integral, (6.52), can then be written as an integral over the unit sphere

$$I = abc \int_0^{2\pi} \int_0^{\pi} \int_0^{1} e^{i\mathbf{K}\cdot\mathbf{r}'} r^2 \sin\theta dr d\theta d\phi \tag{6.76}$$

Next, transfer the coefficients (a, b, c) that are in $\mathbf{r}'$ to a new wave vector difference, $\mathbf{K}'$, so that, $(K'_x, K'_y, K'_z) = (aK_x, bK_y, cK_z)$. The position vector returns to its unstretched form and we have

$$I = abc \int_0^{2\pi} \int_0^{\pi} \int_0^{1} e^{i\mathbf{K}'\cdot\mathbf{r}} r^2 \sin\theta dr d\theta d\phi \tag{6.77}$$

Using the results of Section 6.6.3, (6.72), the volume phase integral over an ellipsoid is

$$I = 4\pi abc \frac{\sin(K') - K'\cos(K')}{K'^3} \tag{6.78}$$

where K' is the magnitude of $\mathbf{K}'$. The volume of an ellipsoid is $V = 4/3\pi abc$, and this can be written

$$I = V \frac{3(\sin(K') - K'\cos(K'))}{K'^3} \tag{6.79}$$

Chapter 7

T-Matrix

The transition matrix, or T-matrix, relates the multipole coefficients of the scattered field of an object to the multipole coefficients of the field incident on the object. The method was pioneered by Waterman [32]. A T-matrix can represent the scattering solution of a single object or a collection of objects, [2]. Deriving or computing the elements of a T-matrix is its own subject, but simple objects admit analytic solutions. This chapter gives the basic definition of the T-matrix and instructions for rotation. Then equations and routines for the T-matrix elements of dielectric and metal spheres are provided. Derivations and routines are then provided to convert between a T-matrix and an S-matrix. Finally, we give expressions for different radar cross sections in terms of the T-matrix and evaluate these for spheres.

7.1 Definition

The incident field is expanded in regular spherical wave functions around the object, and the scattered field is expanded in radiating wave functions. The harmonic sums are written in matrix notation as

$$\mathbf{E}_{inc}(\mathbf{r}) = \begin{bmatrix} Rg\mathbf{M}^t(k,\mathbf{r}) & Rg\mathbf{N}^t(k,\mathbf{r}) \end{bmatrix} \begin{bmatrix} \mathbf{a} \\ \mathbf{b} \end{bmatrix} \tag{7.1}$$

$$\mathbf{E}_{sca}(\mathbf{r}) = \begin{bmatrix} \mathbf{M}^t(k,\mathbf{r}) & \mathbf{N}^t(k,\mathbf{r}) \end{bmatrix} \begin{bmatrix} \mathbf{c} \\ \mathbf{d} \end{bmatrix} \tag{7.2}$$

The expression for the incident field is valid everywhere, while the scattered field is only valid outside the smallest radius that encloses the object or collection of objects. In other words, a T-matrix does not model the scattering solution within an object region. Next, define the sub-matrices that relate the expansion coefficients between the $\mathbf{M}$ and $\mathbf{N}$ harmonics as

$$\begin{bmatrix} \mathbf{c} \\ \mathbf{d} \end{bmatrix} = \begin{bmatrix} \overline{\mathbf{T}}^{MM} & \overline{\mathbf{T}}^{MN} \\ \overline{\mathbf{T}}^{NM} & \overline{\mathbf{T}}^{NN} \end{bmatrix} \begin{bmatrix} \mathbf{a} \\ \mathbf{b} \end{bmatrix} \tag{7.3}$$

An object and its T-matrix can be rotated using the rotation matrix for spherical harmonics. Given an object T-matrix, the T-matrix of the object rotated by ZXZ Euler angles (α, β, γ) is

$$\begin{bmatrix} \overline{\mathbf{T}}'^{MM} & \overline{\mathbf{T}}'^{MN} \\ \overline{\mathbf{T}}'^{NM} & \overline{\mathbf{T}}'^{NN} \end{bmatrix} = \begin{bmatrix} \mathbf{D}^* & 0 \\ 0 & \mathbf{D}^* \end{bmatrix} \begin{bmatrix} \overline{\mathbf{T}}^{MM} & \overline{\mathbf{T}}^{MN} \\ \overline{\mathbf{T}}^{NM} & \overline{\mathbf{T}}^{NN} \end{bmatrix} \begin{bmatrix} \mathbf{D} & 0 \\ 0 & \mathbf{D} \end{bmatrix} \tag{7.4}$$

where $\mathbf{D}$ describes the inverse rotation and $\overline{\mathbf{T}}'$ is the T-matrix of the rotated object as seen from the fixed frame. The field first viewed from the frame of the rotated object, then the T-matrix is applied in its natural frame, last the perspective is rotated back to the fixed frame (an inverse rotation in the opposite direction, i.e., a forward rotation, $\mathbf{D}^*$.)

7.2 Object T-matrix

This section derives the T-matrix for several simple objects.

7.2.1 Dielectric Sphere

Let a sphere of radius a be centered at the origin with outer and inner regions having wavenumbers k_1 and k_2, respectively. The incident, scattered, and total electric fields are expanded in terms of vector spherical wave functions. The regular form of the wave functions is used for the incident everywhere and the total field inside the sphere, while the singular form is used for the scattered field outside of the sphere.

$$\mathbf{E}_{inc}(\mathbf{r}) = \begin{bmatrix} Rg\mathbf{M}^t(k_1,\mathbf{r}) & Rg\mathbf{N}^t(k_1,\mathbf{r}) \end{bmatrix} \begin{bmatrix} \mathbf{a} \\ \mathbf{b} \end{bmatrix}, \quad \forall\ \mathbf{r} \tag{7.5}$$

$$\mathbf{E}_{sca}(\mathbf{r}) = \begin{bmatrix} \mathbf{M}^t(k_1,\mathbf{r}) & \mathbf{N}^t(k_1,\mathbf{r}) \end{bmatrix} \begin{bmatrix} \mathbf{c} \\ \mathbf{d} \end{bmatrix}, \quad \mathbf{r} \geq a \tag{7.6}$$

$$\mathbf{E}_{tot}(\mathbf{r}) = \begin{bmatrix} Rg\mathbf{M}^t(k_2,\mathbf{r}) & Rg\mathbf{N}^t(k_2,\mathbf{r}) \end{bmatrix} \begin{bmatrix} \mathbf{e} \\ \mathbf{f} \end{bmatrix}, \quad \mathbf{r} \leq a \tag{7.7}$$

The magnetic fields are found by taking the curl of each expression (i.e., Faraday's law) and applying the curl relations for vector spherical wave functions. For example, the scattered magnetic field is

$$\mathbf{H}_{sca}(\mathbf{r}) = \frac{k}{i\omega\mu} \begin{bmatrix} \mathbf{N}^t(k,\mathbf{r}) & \mathbf{M}^t(k,\mathbf{r}) \end{bmatrix} \begin{bmatrix} \mathbf{c} \\ \mathbf{d} \end{bmatrix} \tag{7.8}$$

Boundary conditions require that the tangential electric and magnetic fields at $r = a$ must be equal

$$\hat{\mathbf{r}} \times \mathbf{E}_{tot} = \hat{\mathbf{r}} \times \mathbf{E}_{inc} + \hat{\mathbf{r}} \times \mathbf{E}_{sca} \tag{7.9}$$

$$\hat{\mathbf{r}} \times \mathbf{H}_{tot} = \hat{\mathbf{r}} \times \mathbf{H}_{inc} + \hat{\mathbf{r}} \times \mathbf{H}_{sca} \tag{7.10}$$

We want to express the total and scattered field coefficients in terms of the incident field coefficients. We outline the steps and state the result. Left dot (7.9) and (7.10) by $\hat{\mathbf{M}}(k_1,\mathbf{r})$ and by $\hat{\mathbf{N}}(k_1,\mathbf{r})$, respectively, where the caret, $\hat{}$, means the angular harmonics are conjugated. Apply the vector identity that exchanges dot and cross product, evaluate at $r = a$, and integrate over the unit sphere. Orthogonality will pick out the l harmonics. Solve the resulting system of equations, and after some simplifications, including use of the Wronskian, we get the following relations between the coefficients.

$$c_{lm} = -a_{lm} \frac{\mu_2 j_l(k_2 a)\left[k_1 a j_l(k_1 a)\right]' - \mu_1 j_l(k_1 a)\left[k_2 a j_l(k_2 a)\right]'}{\mu_2 j_l(k_2 a)[k_1 a h_l^{(1)}(k_1 a)]' - \mu_1 h_l^{(1)}(k_1 a)\left[k_2 a j_l(k_2 a)\right]'} \tag{7.11}$$

$$d_{lm} = -b_{lm} \frac{\epsilon_2 j_l(k_2 a)\left[k_1 a j_l(k_1 a)\right]' - \epsilon_1 j_l(k_1 a)\left[k_2 a j_l(k_2 a)\right]'}{\epsilon_2 j_l(k_2 a)[k_1 a h_l^{(1)}(k_1 a)]' - \epsilon_1 h_l^{(1)}(k_1 a)\left[k_2 a j_l(k_2 a)\right]'} \tag{7.12}$$

$$e_{lm} = a_{lm} \frac{i\mu_1}{k_1 a} \frac{1}{\mu_2 j_l(k_2 a)[k_1 a h_l^{(1)}(k_1 a)]' - \mu_1 h_l^{(1)}(k_1 a)\left[k_2 a j_l(k_2 a)\right]'} \tag{7.13}$$

$$f_{lm} = b_{lm} \frac{i\epsilon_2}{k_2 a} \frac{1}{\epsilon_2 j_l(k_2 a)[k_1 a h_l^{(1)}(k_1 a)]' - \epsilon_1 h_l^{(1)}(k_1 a)\left[k_2 a j_l(k_2 a)\right]'} \tag{7.14}$$

where a is the radius, k_1 is the wavenumber outside the sphere, k_2 is the wavenumber inside the sphere, either given by

$$k = \omega\sqrt{\mu\epsilon} = \frac{\omega}{c}\sqrt{\mu_r \epsilon_{rc}} \tag{7.15}$$

$$\epsilon_{rc} = \epsilon_r + i\frac{\sigma}{\omega\epsilon_o} \tag{7.16}$$

where ω is the natural frequency, c is the speed of light in vacuum, μ is the magnetic permeability, μ_r is the relative permeability, ϵ is the dielectric permittivity, ϵ_{rc} is the relative complex permittivity, ϵ_r is the real part of the relative permittivity, σ is the conductivity, and ϵ_o is the permittivity of free space. The Bessel derivatives apply to the argument $x = ka$ as

$$[xz_l(x)]' = z_l(x) + xz_l'(x) \tag{7.17}$$

Equations (7.11) and (7.12) give the scattered field coefficients, while (7.13) and (7.14) give the coefficients for the total field inside the sphere. Dividing (7.11) and (7.12) by a_{lm} and b_{lm}, respectively, gives the T-matrix elements for the dielectric sphere.

$$T^{MM}_{lmlm} = \frac{c_{lm}}{a_{lm}} \tag{7.18}$$

$$T^{NN}_{lmlm} = \frac{d_{lm}}{b_{lm}} \tag{7.19}$$

$$T^{MN}_{lmlm} = 0 \tag{7.20}$$

$$T^{NM}_{lmlm} = 0 \tag{7.21}$$

This T-matrix is diagonal, so there is no mode-mixing, and the elements are only a function the degree l, which are copied for all m at that degree.

The routine `tmatrixDielectricSphere` gives the T-matrix elements of a dielectric sphere up to maximum degree L all m. It takes as input the maximum degree harmonic L, sphere radius, and wavenumbers and inside and outside the sphere. The permeabilities are optional, but if nonzero, must also be included in the wavenumbers.

```
function [Tmm, Tnn] = tmatrixDielectricSphere(L,a,k1,k2,u1,u2)
% T-matrix of a dielectric sphere
%
% L:        Maximum harmonic degree L
% a:        radius of sphere (m)
% k1:       background wavenumber (1/m)
% k2:       sphere wavenumber (1/m)
% u1:       (optional) background relative permeability (must be included in k1)
% u2:       (optional) sphere relative permeability (must be included in k2)
%
% Tmm:      Nx1 diagonal elements of Tmatrix for Mlm, N = L^2 + 2*L
% Tnn:      Nx1 diagonal elements of Tmatrix for Nlm
%
% Dependencies: lmtable, sbesselj, sbesselh, sbesslejp2, sbesselhp2

if nargin == 4
    u1 = 1;
    u2 = 1;
end
k1a = k1*a;
k2a = k2*a;
l = 1:L;
j1 = sbesselj(l,k1a);
j2 = sbesselj(l,k2a);
h1 = sbesselh(l,k1a);
j1p = sbesseljp2(l,k1a);
j2p = sbesseljp2(l,k2a);
h1p = sbesselhp2(l,k1a);
N1 = u2*j2.*j1p - u1*j1.*j2p;
D1 = u1*h1.*j2p - u2*j2.*h1p;
N2 = u1*(k2a)^2*j2.*j1p - u2*(k1a)^2*j1.*j2p;
D2 = u2*(k1a)^2*h1.*j2p - u1*(k2a)^2*j2.*h1p;
Tmmtmp = N1./D1;
Tnntmp = N2./D2;
tab = lmtable(L);
Tmm = Tmmtmp(tab(:,1)).';
Tnn = Tnntmp(tab(:,1)).';
```

7.2.2 PEC Sphere

The T-matrix for a perfect electrical conductor (PEC) sphere is found by taking $\epsilon_2 \to \infty$ and/or $\mu_2 \to 0$ in (7.18) and (7.19). The result is

$$T^{MM}_{lmlm} = -\frac{j_l(ka)}{h^{(1)}_l(ka)} \tag{7.22}$$

$$T^{NN}_{lmlm} = -\frac{[kaj_l(ka)]'}{[kah^{(1)}_l(ka)]'} \tag{7.23}$$

$$T^{MN}_{lmlm} = 0 \tag{7.24}$$

$$T^{NM}_{lmlm} = 0 \tag{7.25}$$

where k is the background wavenumber. The negative sign is the 180 degree phase flip familiar from 1D scattering from a PEC boundary.

The routine `tmatrixPECSphere` gives the T-matrix elements of a PEC sphere up to maximum degree L all $\pm m$.

```
function [Tmm, Tnn] = tmatrixPECSphere(L,a,k)
% T-matrix of a PEC sphere
%
% L:        Maximum harmonic degree L
% a:        radius of sphere (m)
% k:        background wavenumber (1/m)
% Tmm:      Nx1 diagonal elements of Tmatrix for Mlm, N = L^2 + 2*L
% Tnn:      Nx1 diagonal elements of Tmatrix for Nlm
%
% Dependencies: lmtable, sbesselj, sbesselh, sbesslejp2, sbesselhp2

ka = k*a;
l=1:L;
Tmmtmp = -sbesselj(l,ka)./sbesselh(l,ka);
Tnntmp = -sbesseljp2(l,ka)./sbesselhp2(l,ka);
tab = lmtable(L);
Tmm = Tmmtmp(tab(:,1)).';
Tnn = Tnntmp(tab(:,1)).';
```

7.3 Extended Boundary Condition Method

The extended boundary condition method (EBCM) is a technique to compute the T-matrix of a bounded, homogenous object that has an arbitrary or irregular surface. For example, this has been used to determine the T-matrix of short cylinders in [25]. While analytically generic, the EBCM is usually not accurate for highly elongated objects or objects with large concavities. We outline the derivation, comment on the computation, and provide a routine to compute the solution.

Formulation The EBCM makes use of the extinction theorem to express fields both inside and outside the object in terms of an integral over the surface of the object, [2]. The geometry is shown in Figure 7.1. The extended boundaries are two virtual surfaces, S_1 and S_2, that are contained in the volumes, V_1 and V_2, respectively, but do not intersect the actual surface, S. S_1 and S_2 are used to decide the form of the addition theorem for the dyadic Green's function. The fields tangent to S are expanded in terms of vector spherical wave functions, after which the incoming and outgoing field coefficients can be related and the T-matrix found.

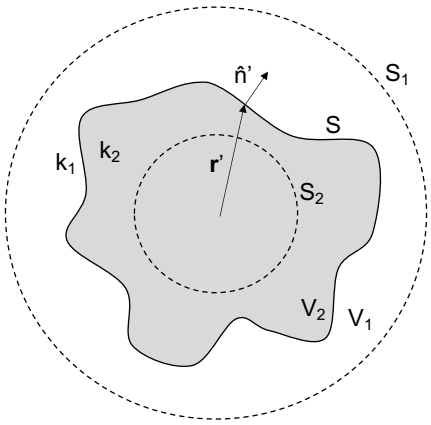


Figure 7.1: Geometry for the Extended Boundary Condition Method.

The surface integral equations used for the EBCM are [2, 25],

$$\mathbf{E}_{inc}(\mathbf{r}) = -\int_S dS' \left(i\omega\mu_1 \overline{\mathbf{G}}_1(\mathbf{r},\mathbf{r}') \cdot \hat{n}' \times \mathbf{H}_1(\mathbf{r}') + \left[\nabla \times \overline{\mathbf{G}}_1(\mathbf{r},\mathbf{r}')\right] \cdot \hat{n}' \times \mathbf{E}_1(\mathbf{r}')\right), \quad \mathbf{r} \in V_2 \tag{7.26}$$

$$0 = -\int_S dS' \left(i\omega\mu_2 \overline{\mathbf{G}}_2(\mathbf{r},\mathbf{r}') \cdot \hat{n}' \times \mathbf{H}_2(\mathbf{r}') + \left[\nabla \times \overline{\mathbf{G}}_2(\mathbf{r},\mathbf{r}')\right] \cdot \hat{n}' \times \mathbf{E}_2(\mathbf{r}')\right), \quad \mathbf{r} \in V_1 \tag{7.27}$$

where $\hat{n}'$ is the outward pointing surface normal (the leading minus sign is missing in [2], and (2.30) has been applied to the curl). The dyadic Green's functions are evaluated at the wavenumbers k_1 and k_2 of the outer and inner regions, respectively. Note, each equation is only valid when the observation point, $\mathbf{r}$, is in the region of the opposite wavenumber.

The addition theorem for the dyadic Green's function, (3.69), is used to expand $\overline{\mathbf{G}}_1(\mathbf{r},\mathbf{r}')$ and $\overline{\mathbf{G}}_2(\mathbf{r},\mathbf{r}')$ on the virtual surfaces, S_2 and S_1, respectively, according the rule that the regular form of the wave functions are used for the smaller of $\mathbf{r}$ or $\mathbf{r}'$. Therefore,

$$\overline{\mathbf{G}}_1(\mathbf{r},\mathbf{r}') = ik_1 \sum_{l=1}^{\infty} \sum_{m=-l}^{l} Rg\mathbf{M}_{lm}(k_1,\mathbf{r})\hat{\mathbf{M}}_{lm}(k_1,\mathbf{r}') + Rg\mathbf{N}_{lm}(k_1,\mathbf{r})\hat{\mathbf{N}}_{lm}(k_1,\mathbf{r}'), \quad \mathbf{r} \in S_2 \tag{7.28}$$

$$\overline{\mathbf{G}}_2(\mathbf{r},\mathbf{r}') = ik_2 \sum_{l=1}^{\infty} \sum_{m=-l}^{l} \mathbf{M}_{lm}(k_2,\mathbf{r})Rg\hat{\mathbf{M}}_{lm}(k_2,\mathbf{r}') + \mathbf{N}_{lm}(k_2,\mathbf{r})Rg\hat{\mathbf{N}}_{lm}(k_2,\mathbf{r}'), \quad \mathbf{r} \in S_1 \tag{7.29}$$

where $\hat{}$ means conjugate of the angular function. The dyad is formed by the outer product of the components of the wave functions, where the left vector is unprimed and the right vector is primed. The vector spherical

wave functions are fully normalized. The curl in (7.26) and (7.27) must be applied to the first vector of the outer product in (7.28) and (7.29), [33], which is the reason for using the unprimed curl. The tangential fields of region 2, $\hat{n}' \times \mathbf{E}_2(\mathbf{r}')$ and $\hat{n}' \times \mathbf{H}_2(\mathbf{r}')$, are expanded as (in matrix notation)

$$\hat{n}' \times \mathbf{E}_2(\mathbf{r}') = \left[\hat{n}' \times Rg\mathbf{M}^t(k_2, \mathbf{r}') \quad \hat{n}' \times Rg\mathbf{N}^t(k_2, \mathbf{r}')\right] \begin{bmatrix} \boldsymbol{\alpha}^M \\ \boldsymbol{\alpha}^N \end{bmatrix} \tag{7.30}$$

$$i\omega\mu_2 \hat{n}' \times \mathbf{H}_2(\mathbf{r}') = \left[\hat{n}' \times \nabla' \times Rg\mathbf{M}^t(k_2, \mathbf{r}') \quad \hat{n}' \times \nabla' \times Rg\mathbf{N}^t(k_2, \mathbf{r}')\right] \begin{bmatrix} \boldsymbol{\beta}^M \\ \boldsymbol{\beta}^N \end{bmatrix} \tag{7.31}$$

Step 1: The first step of EBCM is to derive the relationship between the expansion coefficients $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ subject to the constraint of (7.27). This starts by substituting (7.29) into (7.27). The curl relations for the vector wave functions, (3.65) and (3.66), are applied to the primed wave functions to convert the curl of the wave function into the opposite wave function times the wavenumber. The result is rearranged and expressed as an expansion of unprimed wave functions so that (7.27) becomes

$$0 = \sum_{l=1}^{\infty} \sum_{m=-l}^{l} e_{lm}\mathbf{M}_{lm}(k_2, \mathbf{r}) + f_{lm}\mathbf{N}_{lm}(k_2, \mathbf{r}) \tag{7.32}$$

$$e_{lm} = -ik_2 \int_{S_1} dS' \left[i\omega\mu_2 Rg\hat{\mathbf{M}}_{lm}(k_2, \mathbf{r}') \cdot \hat{n}' \times \mathbf{H}_2(\mathbf{r}') + k_2 Rg\hat{\mathbf{N}}_{lm}(k_2, \mathbf{r}') \cdot \hat{n}' \times \mathbf{E}_2(\mathbf{r}')\right] \tag{7.33}$$

$$f_{lm} = -ik_2 \int_{S_1} dS' \left[i\omega\mu_2 Rg\hat{\mathbf{N}}_{lm}(k_2, \mathbf{r}') \cdot \hat{n}' \times \mathbf{H}_2(\mathbf{r}') + k_2 Rg\hat{\mathbf{M}}_{lm}(k_2, \mathbf{r}') \cdot \hat{n}' \times \mathbf{E}_2(\mathbf{r}')\right] \tag{7.34}$$

Because (7.32) is zero for all points $\mathbf{r} \in V_1$, including points at infinity, and because the wave functions are independent, e_{lm} and f_{lm} have to be equal to zero for all (l, m). Applying the curl relations again to (7.31), then substituting (7.30) and (7.31) into both (7.33) and (7.34), and using a different summation index, we get

$$\begin{aligned} e_{lm} = -ik_2^2 \sum_{p=1}^{\infty} \sum_{q=-p}^{p} \int_{S_1} dS' \Big[& Rg\hat{\mathbf{M}}_{lm}(k_2, \mathbf{r}') \cdot \left(\hat{n}' \times Rg\mathbf{N}_{pq}(k_2, \mathbf{r}')\beta_{pq}^M + \hat{n}' \times Rg\mathbf{M}_{pq}(k_2, \mathbf{r}')\beta_{pq}^N\right) \\ & + Rg\hat{\mathbf{N}}_{lm}(k_2, \mathbf{r}') \cdot \left(\hat{n}' \times Rg\mathbf{M}_{pq}(k_2, \mathbf{r}')\alpha_{pq}^M + \hat{n}' \times Rg\mathbf{N}_{pq}(k_2, \mathbf{r}')\alpha_{pq}^N\right)\Big] \end{aligned} \tag{7.35}$$

$$\begin{aligned} f_{lm} = -ik_2^2 \sum_{p=1}^{\infty} \sum_{q=-p}^{p} \int_{S_1} dS' \Big[& Rg\hat{\mathbf{N}}_{lm}(k_2, \mathbf{r}') \cdot \left(\hat{n}' \times Rg\mathbf{N}_{pq}(k_2, \mathbf{r}')\beta_{pq}^M + \hat{n}' \times Rg\mathbf{M}_{pq}(k_2, \mathbf{r}')\beta_{pq}^N\right) \\ & + Rg\hat{\mathbf{M}}_{lm}(k_2, \mathbf{r}') \left(\hat{n}' \times Rg\mathbf{M}_{pq}(k_2, \mathbf{r}')\alpha_{pq}^M + \hat{n}' \times Rg\mathbf{N}_{pq}(k_2, \mathbf{r}')\alpha_{pq}^N\right)\Big] \end{aligned} \tag{7.36}$$

Using the vector identity $\mathbf{a} \cdot (\mathbf{b} \times \mathbf{c}) = \mathbf{b} \cdot (\mathbf{c} \times \mathbf{a})$, and the fact that cross products of similar wave function are equal to zero, these simplify to

$$e_{lm} = -ik_2^2 \sum_{p=1}^{\infty} \sum_{q=-p}^{p} \int_{S_1} dS' \hat{n}' \cdot \left[Rg\mathbf{N}_{pq}(k_2, \mathbf{r}') \times Rg\hat{\mathbf{M}}_{lm}(k_2, \mathbf{r}')\beta_{pq}^M + Rg\mathbf{M}_{pq}(k_2, \mathbf{r}') \times Rg\hat{\mathbf{N}}_{lm}(k_2, \mathbf{r}')\alpha_{pq}^M\right] \tag{7.37}$$

$$f_{lm} = -ik_2^2 \sum_{p=1}^{\infty} \sum_{q=-p}^{p} \int_{S_1} dS' \hat{n}' \cdot \left[Rg\mathbf{M}_{pq}(k_2, \mathbf{r}') \times Rg\hat{\mathbf{N}}_{lm}(k_2, \mathbf{r}')\beta_{pq}^N + Rg\mathbf{N}_{pq}(k_2, \mathbf{r}') \times Rg\hat{\mathbf{M}}_{lm}(k_2, \mathbf{r}')\alpha_{pq}^N\right] \tag{7.38}$$

Next, choose S_1 to be a sphere so that $\hat{n}' = \hat{r}$. From the cross-product relations for the vector spherical wave functions over a sphere (3.75)-(3.77), the integrals will be non-zero only when $(l, m) = (p, q)$. The Bessel function products that remain are identical between the two indicies, so these can be factored from the difference. Finally, because $\mathbf{M}_{lm} \times \hat{\mathbf{N}}_{lm} = -\hat{\mathbf{N}}_{lm} \times \mathbf{M}_{lm}$, and $\mathbf{N}_{lm} \times \hat{\mathbf{M}}_{lm} = -\hat{\mathbf{M}}_{lm} \times \mathbf{N}_{lm}$, and using the fact that $e_{lm} = 0$ and $f_{lm} = 0$, we have the constraint

$$0 = \beta_{pq}^M - \alpha_{pq}^M \tag{7.39}$$

$$0 = \beta_{pq}^N - \alpha_{pq}^N \tag{7.40}$$

or $\boldsymbol{\alpha}^M = \boldsymbol{\beta}^M$ and $\boldsymbol{\alpha}^N = \boldsymbol{\beta}^N$ for all harmonics.

Step 2: The second step of the EBCM is to use (7.26) to derive a relationship between expansion coefficients of the incident and tangential fields. This starts by expanding the incident field as regular waves in region 2 using the wavenumber of region 1

$$\mathbf{E}_{inc}(\mathbf{r}) = \begin{bmatrix} Rg\mathbf{M}^t(k_1,\mathbf{r}) & Rg\mathbf{N}^t(k_1,\mathbf{r}) \end{bmatrix} \begin{bmatrix} \mathbf{a} \\ \mathbf{b} \end{bmatrix}, \quad \mathbf{r} \in V_2 \tag{7.41}$$

Substituting (7.41) and (7.28) into (7.26), applying the curl, and matching like harmonics of the unprimed wave functions, the incident field expansion coefficients are written in terms of the tangential fields as

$$a_{lm} = -ik_1 \int_S dS' \left(i\omega\mu_1 \hat{\mathbf{M}}_{lm}(k_1,\mathbf{r}') \cdot \hat{n}' \times \mathbf{H}_1(\mathbf{r}') + k_1 \hat{\mathbf{N}}_{lm}(k_1,\mathbf{r}') \cdot \hat{n}' \times \mathbf{E}_1(\mathbf{r}') \right) \tag{7.42}$$

$$b_{lm} = -ik_1 \int_S dS' \left(i\omega\mu_1 \hat{\mathbf{N}}_{lm}(k_1,\mathbf{r}') \cdot \hat{n}' \times \mathbf{H}_1(\mathbf{r}') + k_1 \hat{\mathbf{M}}_{lm}(k_1,\mathbf{r}') \cdot \hat{n}' \times \mathbf{E}_1(\mathbf{r}') \right) \tag{7.43}$$

Because the tangential fields of region 1 and region 2 must be equal on S, then $\hat{n}' \times \mathbf{E}_1(\mathbf{r}')$ and $\hat{n}' \times \mathbf{H}_1(\mathbf{r}')$ can be expanded using (7.30) and (7.31), respectively. The tangential fields of region 1 are then

$$\hat{n}' \times \mathbf{E}_1(\mathbf{r}') = \begin{bmatrix} \hat{n}' \times Rg\mathbf{M}^t(k_2,\mathbf{r}') & \hat{n}' \times Rg\mathbf{N}^t(k_2,\mathbf{r}') \end{bmatrix} \begin{bmatrix} \boldsymbol{\beta}^M \\ \boldsymbol{\beta}^N \end{bmatrix} \tag{7.44}$$

$$\hat{n}' \times \mathbf{H}_1(\mathbf{r}') = \frac{k_2}{i\omega\mu_2} \begin{bmatrix} \hat{n}' \times Rg\mathbf{N}^t(k_2,\mathbf{r}') & \hat{n}' \times Rg\mathbf{M}^t(k_2,\mathbf{r}') \end{bmatrix} \begin{bmatrix} \boldsymbol{\beta}^M \\ \boldsymbol{\beta}^N \end{bmatrix} \tag{7.45}$$

where we have used $\boldsymbol{\alpha} = \boldsymbol{\beta}$ as well as the curl relations. Next, substitute (7.44) and (7.45) into both (7.42) and (7.43). This is done with a separate summation index. Collecting terms, we can write the result in matrix notation as

$$\begin{bmatrix} \mathbf{a} \\ \mathbf{b} \end{bmatrix} = -i \begin{bmatrix} \overline{\mathbf{Q}}^{MM} & \overline{\mathbf{Q}}^{MN} \\ \overline{\mathbf{Q}}^{NM} & \overline{\mathbf{Q}}^{NN} \end{bmatrix} \cdot \begin{bmatrix} \boldsymbol{\beta}^M \\ \boldsymbol{\beta}^N \end{bmatrix} = -i\overline{\mathbf{Q}} \cdot \begin{bmatrix} \boldsymbol{\beta}^M \\ \boldsymbol{\beta}^N \end{bmatrix} \tag{7.46}$$

where

$$\left[\overline{\mathbf{Q}}^{MM}\right]_{lm,pq} = k_1^2 \int_S dS' \left(\left(\frac{\mu_1 k_2}{\mu_2 k_1} \right) \hat{\mathbf{M}}_{lm}(k_1,\mathbf{r}') \cdot \hat{n}' \times Rg\mathbf{N}_{pq}(k_2,\mathbf{r}') + \hat{\mathbf{N}}_{lm}(k_1,\mathbf{r}') \cdot \hat{n}' \times Rg\mathbf{M}_{pq}(k_2,\mathbf{r}') \right) \tag{7.47}$$

$$\left[\overline{\mathbf{Q}}^{MN}\right]_{lm,pq} = k_1^2 \int_S dS' \left(\left(\frac{\mu_1 k_2}{\mu_2 k_1} \right) \hat{\mathbf{M}}_{lm}(k_1,\mathbf{r}') \cdot \hat{n}' \times Rg\mathbf{M}_{pq}(k_2,\mathbf{r}') + \hat{\mathbf{N}}_{lm}(k_1,\mathbf{r}') \cdot \hat{n}' \times Rg\mathbf{N}_{pq}(k_2,\mathbf{r}') \right) \tag{7.48}$$

$$\left[\overline{\mathbf{Q}}^{NM}\right]_{lm,pq} = k_1^2 \int_S dS' \left(\left(\frac{\mu_1 k_2}{\mu_2 k_1} \right) \hat{\mathbf{N}}_{lm}(k_1,\mathbf{r}') \cdot \hat{n}' \times Rg\mathbf{N}_{pq}(k_2,\mathbf{r}') + \hat{\mathbf{M}}_{lm}(k_1,\mathbf{r}') \cdot \hat{n}' \times Rg\mathbf{M}_{pq}(k_2,\mathbf{r}') \right) \tag{7.49}$$

$$\left[\overline{\mathbf{Q}}^{NN}\right]_{lm,pq} = k_1^2 \int_S dS' \left(\left(\frac{\mu_1 k_2}{\mu_2 k_1} \right) \hat{\mathbf{N}}_{lm}(k_1,\mathbf{r}') \cdot \hat{n}' \times Rg\mathbf{M}_{pq}(k_2,\mathbf{r}') + \hat{\mathbf{M}}_{lm}(k_1,\mathbf{r}') \cdot \hat{n}' \times Rg\mathbf{N}_{pq}(k_2,\mathbf{r}') \right) \tag{7.50}$$

Step 3: The third step is to relate scattered field coefficients to the incident field coefficients. For this, the surface integral equation for the scattered field in region 1 is given by, [2],

$$\mathbf{E}_{sca}(\mathbf{r}) = \int_S dS' \left(i\omega\mu_1 \overline{\mathbf{G}}_1(\mathbf{r},\mathbf{r}') \cdot \hat{n}' \times \mathbf{H}_1(\mathbf{r}') + \left[\nabla \times \overline{\mathbf{G}}_1(\mathbf{r},\mathbf{r}')\right] \cdot \hat{n}' \times \mathbf{E}_1(\mathbf{r}') \right), \quad \mathbf{r} \in V_1 \tag{7.51}$$

In the extinction theorem, the surface integral equations for each Green's function come in pairs, and (7.51) is the second of the pair for $\overline{\mathbf{G}}_1(\mathbf{r},\mathbf{r}')$. The scattered field is expanded with radiating waves in region

1 as

$$\mathbf{E}_{sca}(\mathbf{r}) = \begin{bmatrix}\mathbf{M}^t(k_1,\mathbf{r}) & \mathbf{N}^t(k_1,\mathbf{r})\end{bmatrix}\begin{bmatrix}\mathbf{c}\\ \mathbf{d}\end{bmatrix}, \quad \mathbf{r}\in V_1 \tag{7.52}$$

Using the form of the dyadic Green's function in (7.29), but evaluated with k_1, and then repeating the procedure of Step 2, it can be shown that

$$\begin{bmatrix}\mathbf{c}\\ \mathbf{d}\end{bmatrix} = iRg\overline{\mathbf{Q}}\cdot\begin{bmatrix}\boldsymbol{\beta}^M\\ \boldsymbol{\beta}^N\end{bmatrix} \tag{7.53}$$

where $Rg\overline{\mathbf{Q}}$ means use the regular form of the vector wave functions with (l,m) index that are contributed by the addition theorem.

Step 4: Finally, solving (7.46) for $\boldsymbol{\beta}$, substituting the result into (7.53), and applying the definition of the T-matrix, (7.3), we get

$$\overline{\mathbf{T}} = -Rg\overline{\mathbf{Q}}\cdot\overline{\mathbf{Q}}^{-1} \tag{7.54}$$

The T-matrix is independent the wave function normalization. Our derivation uses fully normalized wave functions. The derivation in [25] used partially normalized wave functions were a factor of $1/l(l+1)$ appears in the Green's function addition theorem and therefore in the expressions for the $\overline{\mathbf{Q}}$ matrices. However, a factor of $1/\sqrt{l(l+1)}$ can be distributed to each wave function to make them fully normalized.

We can rederive the T-matrix of the dielectric sphere in Section 7.2.1 to validate (7.47)-(7.50). From (3.75)-(3.77), the cross products of wave functions in (7.47)-(7.50), when integrated over the surface of the sphere, will be zero if the pair of wave functions are the same type, and will be nonzero for dissimilar pairs. From orthogonality, only like-harmonics survive the integration. These mean that $\overline{\mathbf{Q}}^{MM}$ and $\overline{\mathbf{Q}}^{NN}$ will be diagonal and $\overline{\mathbf{Q}}^{MN} = \overline{\mathbf{Q}}^{NM} = 0$. From (3.77), what remains are combinations of products of spherical Bessel functions, spherical Hankel functions, and their modified derivatives. The fact that the cross products in (7.47) or (7.50) are flipped yields a minus sign so that we get a difference between products of Bessel functions. Finally, $Rg\overline{\mathbf{Q}}$ will put regular-type Bessel functions in the numerator of the T-matrix diagonal, while the inverse, $\overline{\mathbf{Q}}^{-1}$, will put mixed Bessel-Hankel products in the denominator. This is precisely the result in (7.11) and (7.12).

Computation Notes on the computation:

- The surface integrals in (7.47)-(7.50) have to be discretized. Each surface point will have its own differential surface area and surface normal unit vector.
- The infinite sums of the expansions have to be truncated. The maximum degree harmonic, L, must be large enough to keep the solution accurate. However, the amplitude of the T-matrix elements decay beyond a certain maximum harmonic based on the object size and wavelength. In practice, the computation can be performed multiple times with finer surface discretization and more harmonics until the T-matrix coverages to a stable value.
- $\overline{\mathbf{Q}}$ is a square matrix. This facilitates the matrix inverse and yields a square T-matrix.
- Direct inversion of $\overline{\mathbf{Q}}^{-1}$ is not recommended. Use matrix decomposition or indirect inversion.
- Both $Rg\overline{\mathbf{Q}}$ and $\overline{\mathbf{Q}}$ need to be stored in order to compute the matrix inverse and matrix multiplication. They also have to be stored separately because they contain different mixtures of radial Bessel functions.
- There are only four unique wave function cross products in (7.47)-(7.50).
- The constant k_1^2 in (7.47)-(7.50) can be ignored because it will cancel in (7.54).
- Because $\overline{\mathbf{Q}}$ must be stored, and because our wave function routines, `BC` and `MN`, return all harmonics for a given set of points, it is advantageous to compute $Rg\overline{\mathbf{Q}}$ and $\overline{\mathbf{Q}}$ as a running sum over the surface discretization. This is likely slower than building and manipulating temporary 3D arrays spanning the two harmonics indices and all surface points. However, trading storage for computation time allows the surface discretization to be made arbitrarily small for a fixed number of harmonics.

- The wave functions are separable in spherical coordinates, the radial functions only differ in wavenumber, and the angular functions only differ by conjugation. It is advantageous to deconstruct, and then rebuild, the wave functions combinations inline.
- The four submatrices, (7.47)-(7.50), can be thought of as being formed by the outer product between the harmonic indices (l,m) and (p,q), where (l,m) are along rows and (p,q) are along columns. For any surface point, we only need to compute two sets of 1D arrays that contain harmonics up to a maximum degree L and then construct the outer product.
- The vector identity $\mathbf{a}\cdot(\mathbf{b}\times\mathbf{c})=\mathbf{b}\cdot(\mathbf{c}\times\mathbf{a})$ can be used in (7.47)-(7.50) to bring the surface normal outside the parentheses and write the matrix elements as cross products of wave functions. In doing so, the vector products can be computed once and then applied to all harmonics as simple scale factors.
- Finally, we use the following shorthand and derive simplifications for wave function cross product combinations. This assumes that the computation loops over surface points and constructs the four submatrices as outer products of harmonics per surface point. Let the $\mathbf{M}$ and $\mathbf{N}$ wave functions, Section 3.6, be written in the shorthand

$$\mathbf{M}_j = c_j\mathbf{C}_j \tag{7.55}$$
$$\mathbf{N}_j = p_j\mathbf{P}_j + b_j\mathbf{B}_j \tag{7.56}$$

where $c_j = c(k_j r)$, $p_j = p(k_j r)$, and $b_j = b(k_j r)$ are the regular or irregular radial Bessel functions associated with each vector spherical harmonic function. The index $j=[1,2]$ keeps track of the wavenumber of the Bessel functions, k_1 or k_2, as well as the family of index: (l,m) or (p,q). From Section 3.5, we can write the vector spherical harmonics in the shorthand

$$\mathbf{P}_j = W_j\hat{r} \tag{7.57}$$
$$\mathbf{B}_j = V_j\hat{\theta} + U_j\hat{\phi} \tag{7.58}$$
$$\mathbf{C}_j = U_j\hat{\theta} - V_j\hat{\phi} \tag{7.59}$$

where U, V, W are the unique spherical functions that make up the vector spherical harmonics. Next, we need the following cross products

$$\mathbf{C}_1\times\mathbf{C}_2 = \hat{r}(V_1U_2 - U_1V_2) \tag{7.60}$$
$$\mathbf{B}_1\times\mathbf{B}_2 = \hat{r}(V_1U_2 - U_1V_2) \tag{7.61}$$
$$\mathbf{C}_1\times\mathbf{B}_2 = \hat{r}(U_1U_2 + V_1V_2) \tag{7.62}$$
$$\mathbf{C}_1\times\mathbf{P}_2 = -(V_1\hat{\theta} + U_1\hat{\phi})W_2 \tag{7.63}$$
$$\mathbf{B}_1\times\mathbf{P}_2 = (U_1\hat{\theta} - V_1\hat{\phi})W_2 \tag{7.64}$$
$$\mathbf{P}_1\times\mathbf{P}_2 = 0 \tag{7.65}$$

Assume that every cross product in (7.47)-(7.50) is converted to a form, for example, $\mathbf{M}_1\cdot(\hat{n}\times\mathbf{N}_2) = -\hat{n}\cdot(\mathbf{M}_1\times\mathbf{N}_2)$. The minus sign can be ignored because it is common to all terms and will cancel in (7.54). The wave function cross products are then:

$$\mathbf{M}_1\times\mathbf{M}_2 = (c_1\mathbf{C_1})\times(c_2\mathbf{C_2}) \tag{7.66}$$
$$= \hat{r}c_1c_2(V_1U_2 - U_1V_2) \tag{7.67}$$
$$\mathbf{N}_1\times\mathbf{N}_2 = (p_1\mathbf{P}_1 + b_1\mathbf{B}_1)\times(p_2\mathbf{P}_2 + b_2\mathbf{B}_2) \tag{7.68}$$
$$= \hat{r}b_1b_2(V_1U_2 - U_1V_2) + \hat{\theta}(b_1U_1p_2W_2 - p_1W_1b_2U_2) + \hat{\phi}(p_1W_1b_2V_2 - b_1V_1p_2W_2) \tag{7.69}$$
$$\mathbf{M}_1\times\mathbf{N}_2 = (c_1\mathbf{C}_1)\times(p_2\mathbf{P}_2 + b_2\mathbf{B}_2) \tag{7.70}$$
$$= \hat{r}c_1b_2(U_1U_2 + V_1V_2) - \hat{\theta}c_1V_1p_2W_2 - \hat{\phi}c_1U_1p_2W_2 \tag{7.71}$$
$$\mathbf{N}_1\times\mathbf{M}_2 = (p_1\mathbf{P}_1 + b_1\mathbf{B}_1)\times(c_2\mathbf{C}_2) \tag{7.72}$$
$$= -\hat{r}b_1c_2(U_1U_2 + V_1V_2) + \hat{\theta}p_1W_1c_2V_2 + \hat{\phi}p_1W_1c_2U_2 \tag{7.73}$$

The dot product between $\hat{n}$ and any unit vector that results from the cross products reduces to a scale factor. Radial and angular functions with like index must to be multiplied together. The product

between functions of index 1 and 2 are computed as an outer product over harmonics. The first wave function always has $\hat{}$, and the Bessel functions of the second wave function are always regular. The vector spherical harmonics between index 1 and 2 only differ by conjugation.
For example, in (7.67), for a single point of the surface discretization, we will a) compute the column vector $f_1 = c_1 U_1$ over (l, m), b) multiply this by any scale factors including the dot product with $\hat{n}$, c) compute the column vector over (p, q) as $f_2 = c_2 U_2 = c_2 U_1^*$, where * means conjugate, d) take the outer product $f_1 \cdot f_2^t$, where ^{t} is transpose, then e) add the resulting matrix to the running sum of $\overline{\mathbf{Q}}$. This is done for all the combinations of wave functions needed in (7.47)-(7.50).

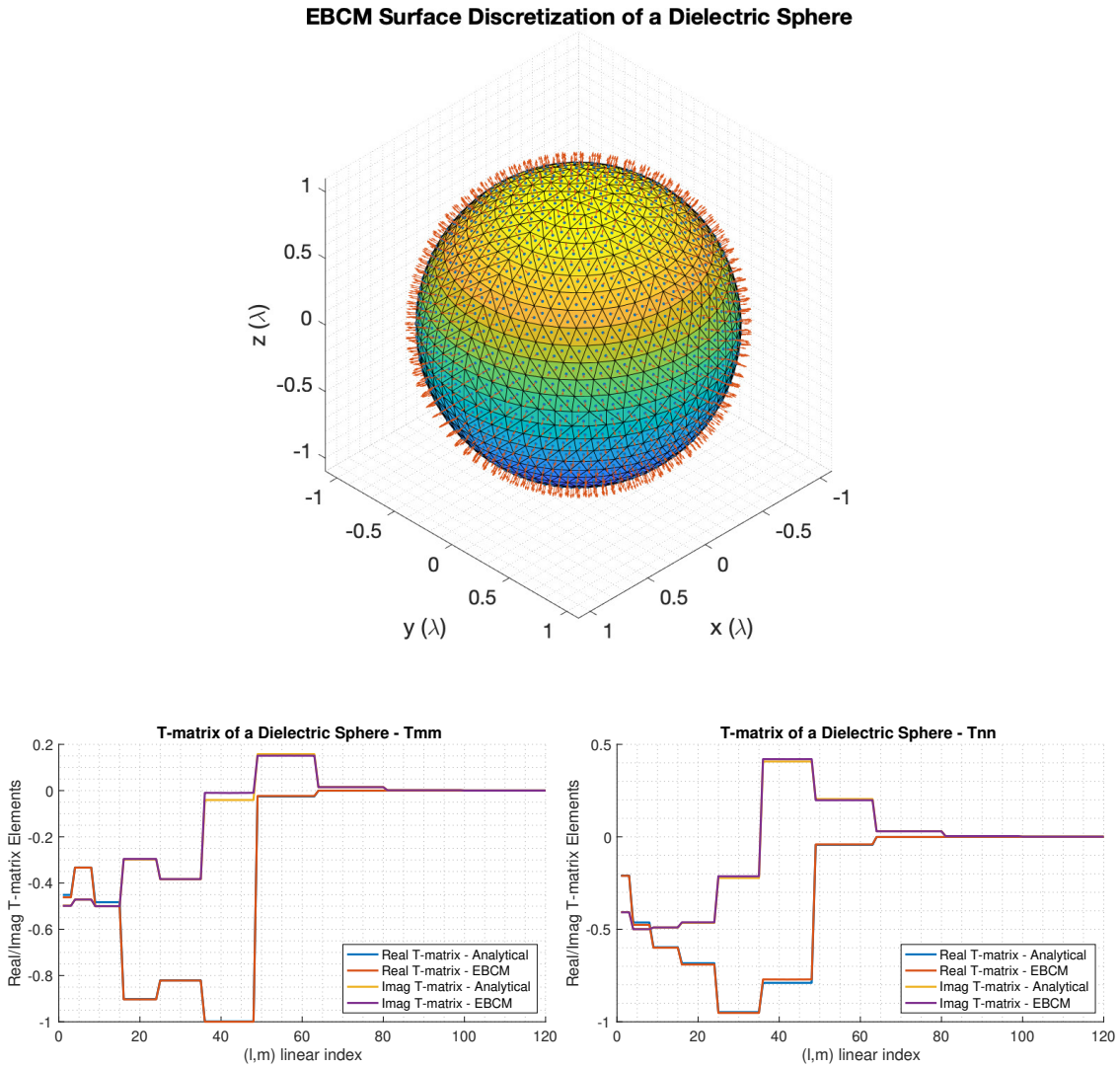


Figure 7.2: Top: Surface discretization of a dielectric sphere for the EBCM. Sphere has a radius of 1λ and a relative permittivity of 2 in a background of free-space. The facet vertices are distributed in a disco ball arraignment. A total of 2192 surface facets are determined from the convex hull of a 3D Delaunay triangularization. Surface normal vectors are shown at the centroid of each facet. The facet centroids are the integration points, $\mathbf{r}'$. Facet area is determined from the cross product of two triangle edge vectors. Bottom: Real and imaginary parts of the diagonal T^{MM} and T^{NN} T-matrices computed with `ebcm` and compared to the T-matrices computed analytically with `tmatrixDielectricSphere`. Small differences are apparent. The EBCM result improves with finer surface discretization.

Routine The routine `ebcm` returns the four block T-matrices computed using the EBCM, (7.54). The T-matrices are square and have harmonics up to maximum degree L all $\pm m$ linearly indexed. The routine takes as input the wavenumbers, k_1 and k_2, in the outer and inner regions, respectively, the maximum degree harmonic, L, and user-provided surface discretization. The surface discretization needs the Cartesian coordinates of surface points, $\mathbf{r}'$, the differential area at each point, dS', and the Cartesian components of the outward surface unit normal vector, $\hat{n}'$, at each point. The arrays describing the surface can be any size, but must be the same size. The permeabilities, μ_1 and μ_2, are optional, and default to 1, and when needed, they can be relative or absolute, but must also be included in k_1 and k_2. The matrices $Rg\overline{\mathbf{Q}}$ and $\overline{\mathbf{Q}}$ are computed as running sums over surface points in order to limit the total size of temporary arrays. The matrix inverse, $\overline{\mathbf{Q}}^{-1}$, is computed with Matlab's right matrix divide, `'/'`. Two helper functions, `ebcmbessel` and `ebcmprod`, are used to organize the code and compute combinations of spherical Bessel and angular functions. The results match the analytic expressions derived for the T-matrix of a dielectric sphere, shown in Figure 7.2.

```
function [Tmm Tmn Tnm Tnn] = ebcm(L,X,Y,Z,dS,nx,ny,nz,k1,k2,mu1,mu2)
% T-matrix computed with extended boundary condition method
%
% L:        Maximum degree harmonic
% X,Y,Z:    [any size, same size] Cartesian points of surface discritization
% dS:       [any size, same as X,Y,Z] Differential surface area for each surface point
% nx,ny,nz: [any size, same as X,Y,Z] Outward surface normal unit vector
% k1,k2:    Wavenumbers in outer and inner regions
% mu1,mu2:  [optional] Permeability of outer and inner regions, realative or absolute, default 1
%
% Tmm, Tmn, Tnm, Tnn:   [NxN] Block T-matrices, N = L^2 + 2L
%
% Dependencies: lmtable,cart2sph,sph2cart,BC,sphericalY,ebcmbessel,ebcmprod

if nargin == 10
    mu1 = 1;
    mu2 = 1;
end

% total number of surface points and harmonics
N = numel(X);
tot = L^2 + 2*L;

% storage for Q matrices
Q = zeros(2*tot,2*tot);
RgQ = zeros(2*tot,2*tot);
ZZ = zeros(tot,tot);
Qmm = ZZ;
Qmn = ZZ;
Qnm = ZZ;
Qnn = ZZ;
RgQmm = ZZ;
RgQmn = ZZ;
RgQnm = ZZ;
RgQnn = ZZ;

% subindices of M and N blocks in Q and RgQ
indM = 1:tot;
indN = (tot+1):(2*tot);

% precompute harmonic linear index
[tab] = lmtable(L);
l = tab(:,1);
l2 = (1:L)';

% multiplying constant
const = mu1*k2/(mu2*k1);

% create n_hat for dot products later
n_hat = [nx(:) ny(:) nz(:)];

% loop over surface points (slow but saves memory)
for n=1:N,
```

```
    % convert surface point to spherical coordinates
    [r th ph] = cart2sph(X(n),Y(n),Z(n));

    % convert spherical unit vectors at this point to Cartesian components
    [r_hat_x r_hat_y r_hat_z]      = sph2cart(r,th,ph,1,0,0);
    [th_hat_x th_hat_y th_hat_z]   = sph2cart(r,th,ph,0,1,0);
    [ph_hat_x ph_hat_y ph_hat_z]   = sph2cart(r,th,ph,0,0,1);

    % dot product of surface normal and spherical unit vectors at the same point
    % multiply by the differential surface element here
    n_dot_r  = dS(n)*dot(n_hat(n,:),[r_hat_x r_hat_y r_hat_z],2);
    n_dot_th = dS(n)*dot(n_hat(n,:),[th_hat_x th_hat_y th_hat_z],2);
    n_dot_ph = dS(n)*dot(n_hat(n,:),[ph_hat_x ph_hat_y ph_hat_z],2);

    % unconjugated vector spherical harmonics up to degree L and +/- m (row arrays)
    [V2, U2, ~, ~] = BC(L,th,ph,'norm');
    W2 = sphericalY(L,th,ph);

    % conjugated vector spherical harmonics (row arrays)
    V1 = conj(V2);
    U1 = conj(U2);
    W1 = conj(W2);

    % bessel functions (row arrays)
    [c1 p1 b1]          = ebcmbessel(tot,L,l,l2,k1,r,[]);
    [rgc1 rgp1 rgb1]    = ebcmbessel(tot,L,l,l2,k1,r,'rg');
    [rgc2 rgp2 rgb2]    = ebcmbessel(tot,L,l,l2,k2,r,'rg');

    % cross product pre-multiplications (column arrays for 1, row arrays for 2)
    [cU1 cV1 bU1 bV1 pW1]           = ebcmprod(c1.',p1.',b1.',U1.',V1.',W1.');
    [rgcU1 rgcV1 rgbU1 rgbV1 rgpW1] = ebcmprod(rgc1.',rgp1.',rgb1.',U1.',V1.',W1.');
    [rgcU2 rgcV2 rgbU2 rgbV2 rgpW2] = ebcmprod(rgc2,rgp2,rgb2,U2,V2,W2);

    % Outer products for Q: (l,m) along rows (index 1), (p,q) along columns (index 2)
    M1hat_cross_rgM2 =  n_dot_r*(cV1*rgcU2 - cU1*rgcV2);
    N1hat_cross_rgN2 =  n_dot_r*(bV1*rgbU2 - bU1*rgbV2) ...
                          + n_dot_th*(bU1*rgpW2 - pW1*rgbU2) + n_dot_ph*(pW1*rgbV2 - bV1*rgpW2);
    M1hat_cross_rgN2 =  n_dot_r*(cU1*rgbU2 + cV1*rgbV2) - n_dot_th*(cV1*rgpW2) - n_dot_ph*(cU1*rgpW2);
    N1hat_cross_rgM2 = -n_dot_r*(bU1*rgcU2 + bV1*rgcV2) + n_dot_th*(pW1*rgcV2) + n_dot_ph*(pW1*rgcU2);

    % Q
    Qmm = Qmm + const*M1hat_cross_rgN2 + N1hat_cross_rgM2;
    Qmn = Qmn + const*M1hat_cross_rgM2 + N1hat_cross_rgN2;
    Qnm = Qnm + const*N1hat_cross_rgN2 + M1hat_cross_rgM2;
    Qnn = Qnn + const*N1hat_cross_rgM2 + M1hat_cross_rgN2;

    % Outer products for RgQ: (l,m) along rows (index 1), (p,q) along columns (index 2)
    rgM1hat_cross_rgM2 =  n_dot_r*(rgcV1*rgcU2 - rgcU1*rgcV2);
    rgN1hat_cross_rgN2 =  n_dot_r*(rgbV1*rgbU2 - rgbU1*rgbV2) ...
                            + n_dot_th*(rgbU1*rgpW2 - rgpW1*rgbU2) + n_dot_ph*(rgpW1*rgbV2 - rgbV1*rgpW2);
    rgM1hat_cross_rgN2 =  n_dot_r*(rgcU1*rgbU2 + rgcV1*rgbV2) - n_dot_th*(rgcV1*rgpW2) - n_dot_ph*(rgcU1*rgpW2);
    rgN1hat_cross_rgM2 = -n_dot_r*(rgbU1*rgcU2 + rgbV1*rgcV2) + n_dot_th*(rgpW1*rgcV2) + n_dot_ph*(rgpW1*rgcU2);

    % RgQ
    RgQmm = RgQmm + const*rgM1hat_cross_rgN2 + rgN1hat_cross_rgM2;
    RgQmn = RgQmn + const*rgM1hat_cross_rgM2 + rgN1hat_cross_rgN2;
    RgQnm = RgQnm + const*rgN1hat_cross_rgN2 + rgM1hat_cross_rgM2;
    RgQnn = RgQnn + const*rgN1hat_cross_rgM2 + rgM1hat_cross_rgN2;

end

% load Q and RgQ
Q(indM,indM) = Qmm;
Q(indM,indN) = Qmn;
Q(indN,indM) = Qnm;
Q(indN,indN) = Qnn;
RgQ(indM,indM) = RgQmm;
RgQ(indM,indN) = RgQmn;
```

```
RgQ(indN,indM) = RgQnm;
RgQ(indN,indN) = RgQnn;

% right matrix divide for matrix inverse to compute T-matrix
T = -RgQ/Q;

% separate block T-matrices
Tmm = T(indM,indM);
Tmn = T(indM,indN);
Tnm = T(indN,indM);
Tnn = T(indN,indN);
end

function [cj pj bj] = ebcmbessel(tot,L,l,l2,k,r,rgstr)
% ebcm helper function to compute Bessel functions
%
% tot:      total number of harmonics
% L:        maximum degree harmonic
% l:        degree l mapped to harmonic linear index (1,1,1,2,2,2,2,2,etc)
% l2:       array of l index 1:L
% k:        wavenumber
% r:        [1x1] radial point
% rgstr:    'rg' for regular form of Bessel functions
%
% cj,pj,bj: [1xtot] radial Bessel functions associated
%           with C,P,B vector spherical harmonics
%
% Dependencies: sbesselj,sbesselj2,sbesseljp,sbesselh,sbesselhp

kr = k*r;
cj = zeros(1,tot);
pj = zeros(1,tot);
bj = zeros(1,tot);
cjtmp = zeros(1,L);
pjtmp = zeros(1,L);
bjtmp = zeros(1,L);

% evalute bessel functions
if strcmp(rgstr,'rg')
    cjtmp(l2) = sbesselj(l2,kr);        % j_l(kr), handles lim r->0
    pjtmp(l2) = sbesselj2(l2,kr);       % j_l(kr)/kr, "
    bjtmp(l2) = sbesseljp(l2,kr);       % j'_l(kr), "
else
    cjtmp(l2) = sbesselh(l2,kr);        % h_l^(1)(kr)
    pjtmp(l2) = sbesselh(l2,kr)./kr;    % h_l^(1)(kr)/kr
    bjtmp(l2) = sbesselhp(l2,kr);       % h'_l^(1)(kr)
end
% complete the B function
bjtmp = pjtmp + bjtmp;

% apply scale factor to P function
pjtmp = sqrt(l2.*(l2+1)).*pjtmp;

% map into the full array
cj(:) = cjtmp(l);
pj(:) = pjtmp(l);
bj(:) = bjtmp(l);
end

function [cU cV bU bV pW] = ebcmprod(c,p,b,U,V,W);
% ebcm helper function to compute wave function components
cU = c.*U;
cV = c.*V;
bU = b.*U;
bV = b.*V;
pW = p.*W;
end
```

7.4 T-matrix to S-matrix Transformation

Here we derive the relationship between the T-matrix and S-matrix. Start with the expression for the scattered field, (7.2), then write out the sums and expand the vector wave functions in their far-field approximations, (3.70) and (3.71),

$$\lim_{kr\to\infty} \mathbf{E}_s(\hat{k}_s) = \frac{e^{ikr}}{kr} \sum_{l=1}^{\infty} \sum_{m=-l}^{l} i^{-l-1} c_{lm} \mathbf{C}_{lm}(\theta_s, \phi_s) + i^{-l} d_{lm} \mathbf{B}_{lm}(\theta_s, \phi_s) \tag{7.74}$$

Substituting the definition of the T-matrix

$$\begin{aligned} \mathbf{E}_s(\hat{k}_s) &= \frac{e^{ikr}}{kr} \sum_{lm} \sum_{l'm'} i^{-l-1} \mathbf{C}_{lm}(\theta_s, \phi_s) \left(T^{MM}_{lm,l'm'} a_{l'm'} + T^{MN}_{lm,l'm'} b_{l'm'} \right) \\ &\quad + i^{-l} \mathbf{B}_{lm}(\theta_s, \phi_s) \left(T^{NM}_{lm,l'm'} a_{l'm'} + T^{NN}_{lm,l'm'} b_{l'm'} \right) \end{aligned} \tag{7.75}$$

The coefficients a_{lm} and b_{lm} are chosen as the expansion coefficients for the vector plane waves, (3.91) and (3.93), giving

$$\begin{aligned} \mathbf{E}_s(\hat{k}_s) &= 4\pi \frac{e^{ikr}}{kr} \sum_{lm} \sum_{l'm'} i^{-l-1} \mathbf{C}_{lm}(\theta_s, \phi_s) \left(T^{MM}_{lm,l'm'} i^{l'} \mathbf{C}^*_{l'm'}(\theta_i, \phi_i) + T^{MN}_{lm,l'm'} i^{l'+1} \mathbf{B}^*_{l'm'}(\theta_i, \phi_i) \right) \cdot \mathbf{E}_i \\ &\quad + i^{-l} \mathbf{B}_{lm}(\theta_s, \phi_s) \left(T^{NM}_{lm,l'm'} i^{l'} \mathbf{C}^*_{l'm'}(\theta_i, \phi_i) + T^{NN}_{lm,l'm'} i^{l'+1} \mathbf{B}^*_{l'm'}(\theta_i, \phi_i) \right) \cdot \mathbf{E}_i \end{aligned} \tag{7.76}$$

where (θ_i, ϕ_i) are the spherical angles of the incident field direction, and $\mathbf{E}_i$ is the polarized incident electric field. At this point, the sum can be computed directly and dotted with the polarization unit vectors as needed. This is written more clearly in matrix notation over the θ, ϕ components of the fields as

$$\begin{aligned} \begin{bmatrix} E_{s,\theta}(\hat{k}_s) \\ E_{s,\phi}(\hat{k}_s) \end{bmatrix} &= 4\pi \frac{e^{ikr}}{kr} \begin{bmatrix} \mathbf{C}^t_\theta(\theta_s, \phi_s) & \mathbf{B}^t_\theta(\theta_s, \phi_s) \\ \mathbf{C}^t_\phi(\theta_s, \phi_s) & \mathbf{B}^t_\phi(\theta_s, \phi_s) \end{bmatrix} \begin{bmatrix} \overline{\mathbf{L}}_1 & 0 \\ 0 & \overline{\mathbf{L}}_2 \end{bmatrix} \begin{bmatrix} \overline{\mathbf{T}}^{MM} & \overline{\mathbf{T}}^{MN} \\ \overline{\mathbf{T}}^{NM} & \overline{\mathbf{T}}^{NN} \end{bmatrix} \\ &\quad \begin{bmatrix} \overline{\mathbf{L}}_2^{-1} & 0 \\ 0 & \overline{\mathbf{L}}_1^{-1} \end{bmatrix} \begin{bmatrix} \mathbf{C}^*_\theta(\theta_i, \phi_i) & \mathbf{C}^*_\phi(\theta_i, \phi_i) \\ \mathbf{B}^*_\theta(\theta_i, \phi_i) & \mathbf{B}^*_\phi(\theta_i, \phi_i) \end{bmatrix} \begin{bmatrix} E_{i,\theta}(\hat{k}_i) \\ E_{i,\phi}(\hat{k}_i) \end{bmatrix} \end{aligned} \tag{7.77}$$

The vector spherical harmonics are column vectors over harmonics (l, m) (where ^{t} is transpose and * is simple conjugate). The coefficient matrices $\mathbf{L}$ are diagonal with elements

$$\left[\overline{\mathbf{L}}_1\right]_{ll} = i^{-l-1} \tag{7.78}$$

$$\left[\overline{\mathbf{L}}_2\right]_{ll} = i^{-l} \tag{7.79}$$

Choosing the orthonormal basis formed by the spherical unit vectors $\hat{k} = \hat{r}$, $\hat{\theta}$, and $\hat{\phi}$, the scattered field can be written

$$\mathbf{E}_s = \left(\hat{\theta} E_{\theta,s} + \hat{\phi} E_{\phi,s} \right) \frac{e^{ikr}}{r} \tag{7.80}$$

From which we have the S-matrix

$$\begin{bmatrix} E_{s,\theta}(\hat{k}_s) \\ E_{s,\phi}(\hat{k}_s) \end{bmatrix} = \begin{bmatrix} S_{\theta\theta}(\hat{k}_s, \hat{k}_i) & S_{\theta\phi}(\hat{k}_s, \hat{k}_i) \\ S_{\phi\theta}(\hat{k}_s, \hat{k}_i) & S_{\phi\phi}(\hat{k}_s, \hat{k}_i) \end{bmatrix} \begin{bmatrix} E_{i,\theta}(\hat{k}_i) \\ E_{i,\phi}(\hat{k}_i) \end{bmatrix} \tag{7.81}$$

Further vectorizing (7.77) over incident and scattered directions, the T-matrix to S-matrix transformation can be written in block matrices, where the block vector spherical harmonics have harmonic indices along rows and wave vector directions along columns. The component S-matrix blocks are

$$\begin{bmatrix} \overline{\mathbf{S}}_{\theta\theta} & \overline{\mathbf{S}}_{\theta\phi} \\ \overline{\mathbf{S}}_{\phi\theta} & \overline{\mathbf{S}}_{\phi\phi} \end{bmatrix} = \frac{4\pi}{k} \begin{bmatrix} \overline{\mathbf{C}}_\theta & \overline{\mathbf{B}}_\theta \\ \overline{\mathbf{C}}_\phi & \overline{\mathbf{B}}_\phi \end{bmatrix} \begin{bmatrix} \overline{\mathbf{L}}_1 & 0 \\ 0 & \overline{\mathbf{L}}_2 \end{bmatrix} \begin{bmatrix} \overline{\mathbf{T}}^{MM} & \overline{\mathbf{T}}^{MN} \\ \overline{\mathbf{T}}^{NM} & \overline{\mathbf{T}}^{NN} \end{bmatrix} \begin{bmatrix} \overline{\mathbf{L}}_2^{-1} & 0 \\ 0 & \overline{\mathbf{L}}_1^{-1} \end{bmatrix} \begin{bmatrix} \overline{\mathbf{C}}^*_\theta & \overline{\mathbf{C}}^*_\phi \\ \overline{\mathbf{B}}^*_\theta & \overline{\mathbf{B}}^*_\phi \end{bmatrix} \tag{7.82}$$

where here * is conjugate transpose. The S-matrix block matrices have scattered field directions along rows, and incident field directions along columns. Recall that the left vector spherical harmonics are evaluated at scattered directions, and the right ones are evaluated at incident directions. This can be used to compute the S-matrix at arbitrary combinations of incident and scattered directions. Also the vector spherical harmonics are fully normalized. See Section 8.8.3 for an exact computation of this using spherical harmonic transforms when the S-matrix can be sampled on the nodes of Gaussian quadrature.

The routine `compute_S_from_T`, computes the four spherical vector components of the S-matrix given the four components of the T-matrix. The block T-matrices are $N \times N$ with harmonics up to degree L all m linearly indexed. The routine takes the incident and scattered directions, which can be different sizes between them. The size of the output S-matrix block matrices is size of an array that is the concatenation of the scattered and incident direction array in that order. The fully normalized vector spherical harmonics are used. The matrix multiplication is fast, but the routine can run out of memory quickly for large L and large number of wave directions. The routine gives identical results to the ones in Section 8.8.3 based on Gaussian quadrature.

```
function [Stt, Stp, Spt, Spp] = compute_S_from_T(Tmm,Tmn,Tnm,Tnn,L,k,theta_s,phi_s,theta_i,phi_i)
% Compute S-matrix points given full T-matrix
%
% Tmm,Tmn,Tnm,Tnn:    T-matrix block matrices
%                     Size [NxN], N = L^2 + 2L
% L:                  Maximum degree harmonic L
% k:                  Background wavenumber
% theta_s,phi_s:      Scatttered directions, radians, any size, sz_s = size(theta_s)
% theta_i,phi_i:      Incident directions, radians, any size, sz_i = size(theta_i)
%
% Stt,Stp,Spt,Spp:    S-matrix block matrices Theta/Phi components,
%                     Size [sz_s sz_i], concatenated sizes of the input
%                     incident and scattered directions
%
% Dependencies:       BC, lmtable

% size of incident direction arrays
sz_i = size(theta_i);
% size of scattered direction arrays
sz_s = size(theta_s);

% total number of incident/scattered directions
Ni = numel(theta_i);
Ns = numel(theta_s);

% reshape the inputs
theta_i = theta_i(:);
phi_i = phi_i(:);
theta_s = theta_s(:);
phi_s = phi_s(:);

% apply coefficient matricies as sparse diagonal matricies to the T-matrix
N = L^2 + 2*L;
tab = lmtable(L);
l = tab(:,1);
l1 = (1i).^(-l-1);
l2 = (1i).^(-l);
diagind = 1:N;
L1 = sparse(diagind,diagind,l1,N,N);
L2 = sparse(diagind,diagind,l2,N,N);
L1inv = sparse(diagind,diagind,1./l1,N,N);
L2inv = sparse(diagind,diagind,1./l2,N,N);
Tmm = L1*Tmm*L2inv;
Tmn = L1*Tmn*L1inv;
Tnm = L2*Tnm*L2inv;
Tnn = L2*Tnn*L1inv;

% incident direction vector spherical harmonics
[Bth_i, Bphi_i, Cth_i, Cphi_i] = BC(L,theta_i,phi_i,'norm');
```

```
% scattered direction vector spherical harmonics
[Bth_s, Bphi_s, Cth_s, Cphi_s] = BC(L,theta_s,phi_s,'norm');

% build the matrices and multiply
Smatrix = (4*pi/k)*([Cth_s Bth_s; Cphi_s Bphi_s]*...
    ([Tmm Tmn; Tnm Tnn]*[Cth_i' Cphi_i'; Bth_i' Bphi_i']));

% indecies for different components in the full matrix
grab_i_theta = 1:Ni;
grab_s_theta = 1:Ns;
grab_i_phi = (Ni+1):(2*Ni);
grab_s_phi = (Ns+1):(2*Ns);

% Extract the four components and reshape
Stt = reshape(Smatrix(grab_s_theta,grab_i_theta),[sz_s sz_i]);
Stp = reshape(Smatrix(grab_s_theta,grab_i_phi),[sz_s sz_i]);
Spt = reshape(Smatrix(grab_s_phi,grab_i_theta),[sz_s sz_i]);
Spp = reshape(Smatrix(grab_s_phi,grab_i_phi),[sz_s sz_i]);
```

7.5 S-matrix to T-matrix Transformation

The S-matrix to T-matrix transformation requires integrating the S-matrix components and vector spherical harmonics over the unit spheres of incident and scattered directions. Let the diagonal matrix $\overline{\mathbf{W}}$ hold the weights of the discretized surface integral over the sphere, where the integral is applied over the sphere of wave vector directions. From orthogonality of the fully normalized vector spherical harmonics, we have the identity

$$\begin{bmatrix} \overline{\mathbf{C}}_\theta^* & \overline{\mathbf{C}}_\phi^* \\ \overline{\mathbf{B}}_\theta^* & \overline{\mathbf{B}}_\phi^* \end{bmatrix} \begin{bmatrix} \overline{\mathbf{W}} & 0 \\ 0 & \overline{\mathbf{W}} \end{bmatrix} \begin{bmatrix} \overline{\mathbf{C}}_\theta & \overline{\mathbf{B}}_\theta \\ \overline{\mathbf{C}}_\phi & \overline{\mathbf{B}}_\phi \end{bmatrix} = \begin{bmatrix} \overline{\mathbf{I}} & 0 \\ 0 & \overline{\mathbf{I}} \end{bmatrix} \tag{7.83}$$

This identity assumes that the weights are chosen so that the integral is computed exactly. Using this, (7.82) can be manipulated to give

$$\begin{bmatrix} \overline{\mathbf{T}}^{MM} & \overline{\mathbf{T}}^{MN} \\ \overline{\mathbf{T}}^{NM} & \overline{\mathbf{T}}^{NN} \end{bmatrix} = \frac{k}{4\pi} \begin{bmatrix} \overline{\mathbf{L}}_1^{-1} & 0 \\ 0 & \overline{\mathbf{L}}_2^{-1} \end{bmatrix} \begin{bmatrix} \overline{\mathbf{C}}_\theta^* & \overline{\mathbf{C}}_\phi^* \\ \overline{\mathbf{B}}_\theta^* & \overline{\mathbf{B}}_\phi^* \end{bmatrix} \begin{bmatrix} \overline{\mathbf{W}} & 0 \\ 0 & \overline{\mathbf{W}} \end{bmatrix} \begin{bmatrix} \overline{\mathbf{S}}_{\theta\theta} & \overline{\mathbf{S}}_{\theta\phi} \\ \overline{\mathbf{S}}_{\phi\theta} & \overline{\mathbf{S}}_{\phi\phi} \end{bmatrix} \begin{bmatrix} \overline{\mathbf{W}} & 0 \\ 0 & \overline{\mathbf{W}} \end{bmatrix} \begin{bmatrix} \overline{\mathbf{C}}_\theta & \overline{\mathbf{B}}_\theta \\ \overline{\mathbf{C}}_\phi & \overline{\mathbf{B}}_\phi \end{bmatrix} \begin{bmatrix} \overline{\mathbf{L}}_2 & 0 \\ 0 & \overline{\mathbf{L}}_1 \end{bmatrix} \tag{7.84}$$

This expression transforms the S-matrix to the T-matrix under the condition that the S-matrix is fully sampled and can be accurately integrated with $\overline{\mathbf{W}}$. See Section 8.8.2 for a fast routine that computes this integral exactly when the S-matrix is sampled at the nodes of Gaussian quadrature.

7.6 Far-field T-matrix

In (7.82) the inner three matrices of the T-matrix to S-matrix transformation convert between vector spherical harmonic expansions on either side of the T-matrix. From this, it is possible to define a far-field T-matrix that transforms between incoming and outgoing far-field plane-wave patterns that are expressed in vector spherical harmonics.

Let an incoming far-field pattern of a field and its vector spherical harmonic expansion be defined

$$\mathbf{F}(\theta,\phi) = F_\theta(\theta,\phi)\hat{\theta} + F_\phi(\theta,\phi)\hat{\phi} \tag{7.85}$$

$$= \sum_{l=1}^{L} \sum_{m=-l}^{l} b_{lm}\mathbf{B}_{lm}(\theta,\phi) + c_{lm}\mathbf{C}_{lm}(\theta,\phi) \tag{7.86}$$

and let an outgoing far-field pattern due to a scatterer and its vector spherical harmonic expansion be

$$\mathbf{F}'(\theta,\phi) = F'_\theta(\theta,\phi)\hat{\theta} + F'_\phi(\theta,\phi)\hat{\phi} \tag{7.87}$$

$$= \sum_{l=1}^{L}\sum_{m=-l}^{l} b'_{lm}\mathbf{B}_{lm}(\theta,\phi) + c'_{lm}\mathbf{C}_{lm}(\theta,\phi) \tag{7.88}$$

Both field expansions are band limited with the same number of harmonics. Define the far-field T-matrix to transform the vector spherical harmonic coefficients of the incoming field to those of the outgoing field:

$$\begin{bmatrix}\mathbf{b}'\\ \mathbf{c}'\end{bmatrix} = \begin{bmatrix}\overline{\mathbf{T}}^{BB} & \overline{\mathbf{T}}^{BC}\\ \overline{\mathbf{T}}^{CB} & \overline{\mathbf{T}}^{CC}\end{bmatrix}\begin{bmatrix}\mathbf{b}\\ \mathbf{c}\end{bmatrix} \tag{7.89}$$

Using (7.82), this is related to the full T-matrix as

$$\begin{bmatrix}\overline{\mathbf{T}}^{BB} & \overline{\mathbf{T}}^{BC}\\ \overline{\mathbf{T}}^{CB} & \overline{\mathbf{T}}^{CC}\end{bmatrix} = \frac{4\pi}{k}\begin{bmatrix}\overline{\mathbf{L}}_2 & 0\\ 0 & \overline{\mathbf{L}}_1\end{bmatrix}\begin{bmatrix}\overline{\mathbf{T}}^{NN} & \overline{\mathbf{T}}^{NM}\\ \overline{\mathbf{T}}^{MN} & \overline{\mathbf{T}}^{MM}\end{bmatrix}\begin{bmatrix}\overline{\mathbf{L}}_1^{-1} & 0\\ 0 & \overline{\mathbf{L}}_2^{-1}\end{bmatrix} \tag{7.90}$$

The reverse is accomplished by simple inversion.

The routines `convert_T_to_Tfar` and `convert_Tfar_to_T` will transform T-matrix to a far-field T-matrix and visa versa. They take as input the four $N \times N$ block matrices, where $N = L^2 + 2L$ and maximum harmonic degree L all m linearly indexed.

```
function [Tbb, Tbc, Tcb, Tcc] = convert_T_to_Tfar(Tmm,Tmn,Tnm,Tnn,L,k)
% Convert T-matrix to far-field T-matrix
%
% Tmm,Tmn,Tnm,Tnn:    T-matrix block matrices
%                     Size [NxN], N = L^2 + 2L
% L:                  Maximum degree harmonic L
% k:                  Background wavenumber
%
% Tbb,Tbc,Tcb,Tcc:    T-matrix block matrices
%                     Size [NxN], N = L^2 + 2L
%
% Dependencies:       lmtable

% apply coefficient matrices as sparse diagonal matrices to the T-matrix
N = L^2 + 2*L;
tab = lmtable(L);
l = tab(:,1);
l1 = (1i).^(-l-1);
l2 = (1i).^(-l);
diagind = 1:N;
L1 = sparse(diagind,diagind,l1,N,N);
L2 = sparse(diagind,diagind,l2,N,N);
L1inv = sparse(diagind,diagind,1./l1,N,N);
L2inv = sparse(diagind,diagind,1./l2,N,N);
const = 4*pi/k;
Tbb = (const*L2)*Tnn*L1inv;
Tbc = (const*L2)*Tnm*L2inv;
Tcb = (const*L1)*Tmn*L1inv;
Tcc = (const*L1)*Tmm*L2inv;


function [Tmm, Tmn, Tnm, Tnn] = convert_Tfar_to_T(Tbb,Tbc,Tcb,Tcc,L,k)
% Convert far-field T-matrix to T-matrix
%
% Tbb,Tbc,Tcb,Tcc:    T-matrix block matrices
%                     Size [NxN], N = L^2 + 2L
% L:                  Maximum degree harmonic L
% k:                  Background wavenumber
%
% Tmm,Tmn,Tnm,Tnn:    T-matrix block matrices
```

```
%                       Size [NxN], N = L^2 + 2L
%
% Dependencies:         lmtable

% apply coefficient matrices as sparse diagonal matrices to the T-matrix
N = L^2 + 2*L;
tab = lmtable(L);
l = tab(:,1);
l1 = (1i).^(-l-1);
l2 = (1i).^(-l);
diagind = 1:N;
L1 = sparse(diagind,diagind,l1,N,N);
L2 = sparse(diagind,diagind,l2,N,N);
L1inv = sparse(diagind,diagind,1./l1,N,N);
L2inv = sparse(diagind,diagind,1./l2,N,N);
const = k/(4*pi);
Tnn = (const*L2inv)*Tbb*L1;
Tnm = (const*L2inv)*Tbc*L2;
Tmn = (const*L1inv)*Tcb*L1;
Tmm = (const*L1inv)*Tcc*L2;
```

7.7 Radar Cross Section from T-matrix

The bistatic radar cross section is defined

$$\sigma_{pq}(\hat{k}_s, \hat{k}_i) = \lim_{r\to\infty} 4\pi r^2 \frac{|\hat{p}\cdot\mathbf{E}_s|^2}{|\hat{q}\cdot\mathbf{E}_i|^2} \tag{7.91}$$

This is used with (7.76) to compute the radar cross section given an object T-matrix. In short, the process removes the scale factors $E_o e^{ikr}/r$ from (7.76) and can then be evaluated given incident/scattered directions and polarizations. Because $\sigma_{pq} = 4\pi|S_{pq}|^2$, (6.17), the routine `compute_S_from_T`, Section 7.4, can be used to compute the S-matrix for $\hat{p} = \hat{\theta}$ and $\hat{q} = \hat{\phi}$. This can then be converted to any radar cross section polarization combination using (6.19).

7.7.1 Backscatter Radar Cross Section of a Sphere

For a sphere, the radar backscatter is independent of the incident direction, and (7.76) can be simplified analytically. Let the incident direction be in the $+\hat{z}$ direction, $(\theta_i, \phi_i) = (0, 0)$, and the scattered direction be $-\hat{z}$ with $(\theta_i, \phi_i) = (\pi, 0)$. Let the incident and scattered electric field have both $\hat{x}$ and $\hat{y}$ components

$$\mathbf{E}_i = E_x\hat{x} + E_y\hat{y} \tag{7.92}$$

For a PEC or dielectric sphere, the T-matrix is diagonal with zero cross terms so that (7.77) reduces to one sum as

$$\mathbf{E}_s(\pi,0) = -4\pi i\frac{e^{ikr}}{kr}\sum_{lm}\mathbf{C}_{lm}(\pi,0)T_{lm}^{MM}\mathbf{C}_{lm}^*(0,0)\cdot\mathbf{E}_i - \mathbf{B}_{lm}(\pi,0)T_{lm}^{NN}\mathbf{B}_{lm}^*(0,0)\cdot\mathbf{E}_i \tag{7.93}$$

Using (3.53), (3.54), (3.55), (3.56), this reduces to

$$\mathbf{E}_s(\pi,0) = \frac{-i}{2}\frac{e^{ikr}}{kr}\sum_l(-1)^l(2l+1)\left[\hat{x}(T_l^{MM}-T_l^{NN})E_x - \hat{y}(T_l^{MM}-T_l^{NN})E_y\right] \tag{7.94}$$

This shows that the scattered field in the backscatter direction of a sphere is polarization independent with no cross-pol. Substituting (7.94) into (7.91) and choosing either polarization, the polarization amplitude will cancel. The backscatter radar cross section of a sphere where only the diagonal elements of the T-matrix are non-zero

$$\sigma = \frac{\pi}{k^2}\left|\sum_{l=1}^{\infty}(-1)^l(2l+1)\left(T_l^{MM}-T_l^{NN}\right)\right|^2 \tag{7.95}$$

7.7.2 Backscatter RCS of a PEC Sphere

Substituting the T-matrix for a PEC sphere into (7.95), the backscatter RCS of a PEC sphere is

$$\sigma_{PEC} = \frac{\pi}{k^2}\left|\sum_{l=1}^{\infty}(-1)^l(2l+1)\left(\frac{j_l(ka)}{h_l^{(1)}(ka)} - \frac{[kaj_l(ka)]'}{[kah_l^{(1)}(ka)]'}\right)\right|^2 \tag{7.96}$$

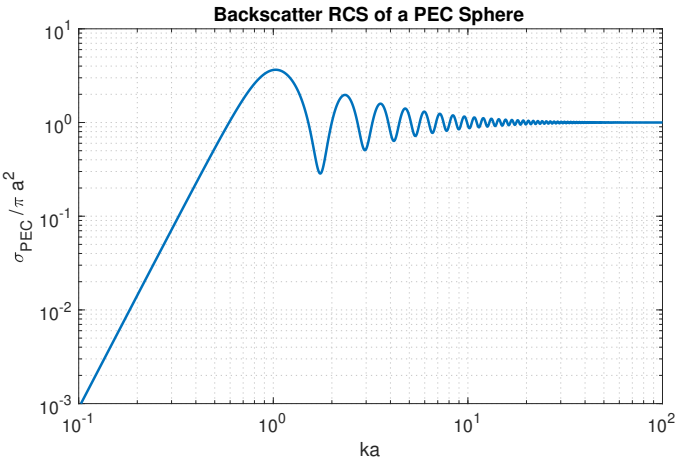


Figure 7.3: RCS of PEC sphere, normalized by cross-section area. The maximum occurs at $ka \approx 1.03$, with a value of about 3.65 (linear).

The routine `brcs_pec_sphere` is a workable routine to compute (7.96). The sum is truncated when incremental change is less than machine precision. The standard and better way to compute this is to use logarithmic derivatives of the Bessel functions, but that is a separate topic.

```
function [brcs] = brcs_pec_sphere(k,a)
% Backscatter Radar Cross Section of a PEC sphere
%
% k:    background wavenumber
% a:    sphere radius
%
% brcs:  Backscatter RCS of the PEC sphere
%
% Dependencies: sbesselj, sbesselh, sbesseljp2, sbesselhp2

ka = k*a;
l = 1;
err = 1;
old = 0;
while err > 1e-12;
    Tmm = sbesselj(l,ka)./sbesselh(l,ka);
    Tnn = sbesseljp2(l,ka)./sbesselhp2(l,ka);
    tmp = (-1).^l.*(2*l+1).*(Tmm - Tnn);
    brcs = old + tmp;
    err = abs(tmp)^2/abs(brcs)^2;
    old = brcs;
    l = l + 1;
end
brcs = pi/(abs(k)^2)*abs(brcs).^2;
```

7.7.3 Backscatter RCS of a Dielectric Sphere

Using the T-matirx for the dielectric sphere in (7.95), the backscatter RCS of a non-magnetic dielectric sphere is

$$\sigma = \frac{\pi}{k^2}\left|\sum_{l=1}^{\infty}(-1)^l(2l+1)\left(\frac{j_2 j_1' - j_1 j_2'}{h_1 j_2' - j_2 h_1'} - \frac{x_2^2 j_2 j_1' - x_1^2 j_1 j_2'}{x_1^2 h_1 j_2' - x_2^2 j_2 h_1'}\right)\right|^2 \tag{7.97}$$

which uses the shorthand:

$$\begin{aligned}
x_1 &= k_1 a && (7.98)\\
x_2 &= k_2 a && (7.99)\\
j_1 &= j_l(k_1 a) && (7.100)\\
j_1' &= [k_1 a j_l(k_1 a)]' && (7.101)\\
j_2 &= j_l(k_2 a) && (7.102)\\
j_2' &= [k_2 a j_l(k_2 a)]' && (7.103)\\
h_1 &= h_l^{(1)}(k_1 a) && (7.104)\\
h_1' &= [k_1 a h_l^{(1)}(k_1 a)]' && (7.105)
\end{aligned}$$

where k_1 is the wavenumber outside the sphere, and k_2 is wavenumber inside the sphere. Making use of the Wronskian $j_l(x)h_l'^{(1)}(x) - h_l^{(1)}(x)j_l'(x) = i/x^2$, it can be shown that $h_1' j_1 - h_1 j_1' = i/x_1$. Then the backscatter can also be written

$$\sigma = \frac{\pi}{k^2}\left|\sum_{l=1}^{\infty}(-1)^l(2l+1)\left(\frac{j_2 j_2'(x_2^2 - x_1^2)}{x_1\,(h_1 j_2' - j_2 h_1')\,(x_1^2 h_1 j_2' - x_2^2 j_2 h_1')}\right)\right|^2 \tag{7.106}$$

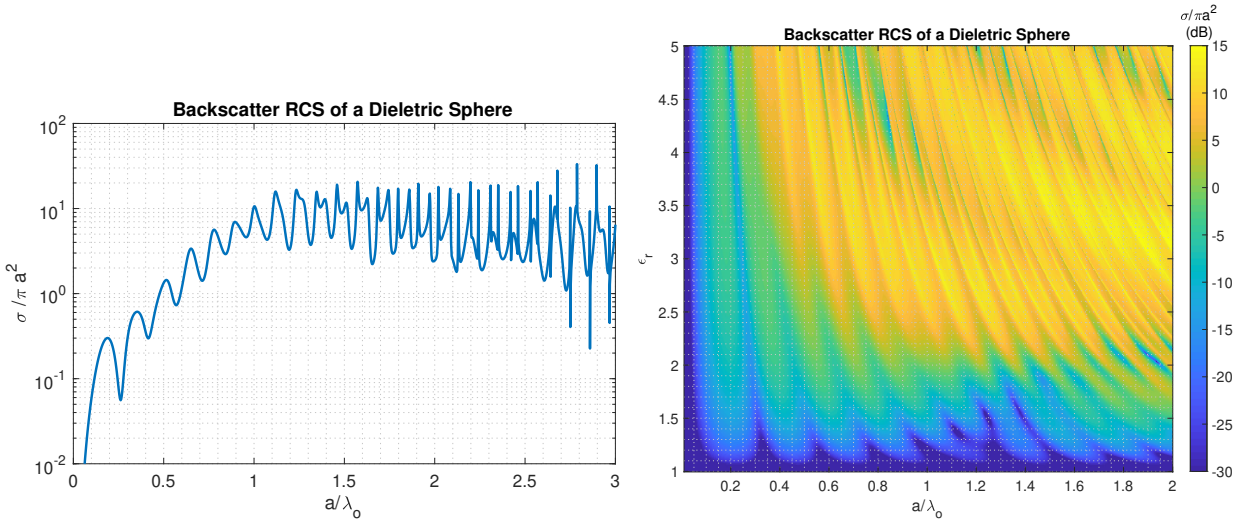


Figure 7.4: Backscatter RCS of dielectric sphere, normalized by cross-section area, in free-space. Left: $\epsilon_r = 2.5$. Right: ϵ_r =1 to 5, and $a/\lambda_o = 0$ to 2.

The routine `brcs_dielectric_sphere` is a workable routine for the computation of (7.97). Assuming a background of vacuum, Figure 7.4(a) shows the backscatter RCS for a sphere with $\epsilon_r = 2.5$ as a function of sphere radius. Figure 7.4(b) shows the same as a function of $k_o a$ and ϵ_r.

```
function [brcs] = brcs_dielectric_sphere(k1,k2,a)
% Backscatter Radar Cross Section of a Dielectric Sphere
%
% k1:   background wavenumber
% k2:   sphere wavenumber
% a:    sphere radius
%
% brcs:  Backscatter RCS of the dielectric sphere
%
% Dependencies: sbesselj, sbesselh, sbesseljp2, sbesselhp2

x1 = k1*a;
x2 = k2*a;
l = 1;
err = 1;
old = 0;
brcs = 0;
while err > eps;
    j2 = sbesselj(l,x2);
    j2p = sbesseljp2(l,x2);
    h1 = sbesselh(l,x1);
    h1p = sbesselhp2(l,x1);
    num = j2.*j2p.*(x2.^2 - x1.^2)./x1;
    t1 = h1.*j2p;
    t2 = j2.*h1p;
    den = (t1 - t2).*(x1.^2.*t1 - x2.^2.*t2);
    tmp = (-1).^l.*(2*l+1).*(num./den);
    brcs = old + tmp;
    err = abs(tmp)^2/abs(brcs)^2;
    old = brcs;
    l = l + 1;
end
brcs = pi/(abs(k1)^2)*abs(brcs).^2;
```

7.8 Scattering Cross Section from T-matrix

The scattering cross section is equal to the scattered power integrated over the sphere for a unit amplitude incident electric field and specific incident direction. From [3], the scattering cross section is written in terms of the S-matrix as

$$\sigma_{s\beta}(\hat{k}_i) = \int_S \left(|S_{p\beta}(\hat{k}_s, \hat{k}_i)|^2 + |S_{q\beta}(\hat{k}_s, \hat{k}_i)|^2 \right) d\Omega_s \tag{7.107}$$

where β is the incident polarization and p and q are any two orthogonal scattered polarizations. Taking the magnitude squared of (7.76) without the terms $E_o e^{ikr}/r$, integrating this over the scattered field directions, and applying orthonormality of the vector spherical harmonics, one gets, similar to [3],

$$\begin{aligned} \sigma_{s\beta}(\hat{k}_i) \quad = \quad & \frac{16\pi^2}{k^2} \sum_{lm} \left\{ \left| \sum_{l'm'} \left(T^{MM}_{lm,l'm'} i^{l'} \mathbf{C}^*_{l'm'}(\theta_i, \phi_i) + T^{MN}_{lm,l'm'} i^{l'+1} \mathbf{B}^*_{l'm'}(\theta_i, \phi_i) \right) \cdot \hat{\beta} \right|^2 \right. \\ & \left. + \left| \sum_{l'm'} \left(T^{NM}_{lm,l'm'} i^{l'} \mathbf{C}^*_{l'm'}(\theta_i, \phi_i) + T^{NN}_{lm,l'm'} i^{l'+1} \mathbf{B}^*_{l'm'}(\theta_i, \phi_i) \right) \cdot \hat{\beta} \right|^2 \right\} \end{aligned} \tag{7.108}$$

The routine `compute_scs_from_tmatrix` computes the scattering cross section of an object from its T-matrix. It takes as input the four $N \times N$ bock T-matrices where $N = L^2 + 2L$, for harmonics up to maximum degree L all m. The incident angles can be any size. The incident polarization is defined relative to the spherical unit vectors $\hat{\beta} = \cos\beta\hat{\theta} + \sin\beta\hat{\phi}$. Recall $\hat{v} = \hat{\theta}$ and $\hat{h} = \hat{\phi}$. Values $\beta = [0, \pi/2]$ correspond to v, h respectively. The array of β can be any size. The scattering cross section is returned on an array with dimensions that are the concatenation of the sizes of the incident direction array and polarization angle array.

```
function [scs] = compute_scs_from_tmatrix(Tmm,Tmn,Tnm,Tnn,L,k,theta_i,phi_i,beta)
% Compute scattering cross section from T-matrix
%
% Tmm,Tmn,Tnm,Tnn:    T-matrix block matrices
%                     Size [NxN], N = L^2 + 2L
% L:                  Maximum degree harmonic L
% k:                  Background wavenumber
% theta_i,phi_i:      Incident directions, radians, any size, sz_i = size(theta_i)
% beta:               Right-hand polarization angle (radians) relative to
%                     \hat{\theta}, any size, sz_b = size(beta)
%
% scs:                Scattering cross section
%                     Size [sz_i sz_b], concatenated sizes of the incident
%                     directions and beta (singular dimensions squeezed)
%
% Dependencies:       BC, lmtable

% size of incident direction arrays and incident directions
sz_i = size(theta_i);
sz_b = size(beta);

% total number of incident directions
Ni = numel(theta_i);
Nb = numel(beta);
scs = zeros(Ni,Nb);

% reshape the inputs
theta_i = theta_i(:);
phi_i = phi_i(:);
beta = beta(:);

% incident direction vector spherical harmonics (sized [Ni x N])
[Bth_i, Bphi_i, Cth_i, Cphi_i] = BC(L,theta_i,phi_i,'norm');

% create coefficient matricies as sparse diagonal matricies
N = L^2 + 2*L;
tab = lmtable(L);
l = tab(:,1);
l1 = (1i).^(l);
l2 = (1i).^(l+1);
diagind = 1:N;
L1 = sparse(diagind,diagind,l1,N,N);
L2 = sparse(diagind,diagind,l2,N,N);

% separate the polarizations, apply conjugate, perform sum of l'm' with
% matrix multiplication. the results are sized [N x Ni].
Th1 = (Tmm*L1)*Cth_i' + (Tmn*L2)*Bth_i';
Ph1 = (Tmm*L1)*Cphi_i' + (Tmn*L2)*Bphi_i';
Th2 = (Tnm*L1)*Cth_i' + (Tnn*L2)*Bth_i';
Ph2 = (Tnm*L1)*Cphi_i' + (Tnn*L2)*Bphi_i';

% for each \beta angle, compute abs^2 and sum over lm
for b=1:Nb,
    tmp1 = sum(abs(Th1*cos(beta(b)) + Ph1*sin(beta(b))).^2,1);
    tmp2 = sum(abs(Th2*cos(beta(b)) + Ph2*sin(beta(b))).^2,1);
    scs(:,b) = tmp1.' + tmp2.';
end

% apply constant
scs = (16*pi^2/(abs(k)^2))*scs;

% reshape and squeeze singular dimensions
scs = squeeze(reshape(scs,[sz_i sz_b]));
```

7.8.1 Scattering Cross Section of a Sphere

For the scattering cross section of a sphere, from symmetry, we only need to consider one incident polarization and an arbitrary incident direction. Assume that the incident plane wave propagates in the z direction:

$$\mathbf{E}_i = E_o \hat{x} \tag{7.109}$$

The T-matrix of the sphere is diagonal with zero cross terms, which means (7.108) simplifies to

$$\sigma_s = \frac{16\pi^2}{k^2} \sum_{lm} \left\{ \left| T^{MM}_{lmlm} \mathbf{C}^*_{lm}(0,0) \cdot \hat{x} \right|^2 + \left| T^{NN}_{lmlm} \mathbf{B}^*_{lm}(0,0) \cdot \hat{x} \right|^2 \right\} \tag{7.110}$$

From (3.53), (3.54), and (7.109), the vector spherical harmonics for z propagation wave have only $m = \pm 1$ harmonics and the dot product with $\hat{x}$ is

$$\mathbf{C}^*_{l,\pm 1}(0,0) \cdot \hat{x} = \frac{i}{2} \sqrt{\frac{2l+1}{4\pi}} \tag{7.111}$$

$$\mathbf{B}^*_{l,\pm 1}(0,0) \cdot \hat{x} = \pm \frac{1}{2} \sqrt{\frac{2l+1}{4\pi}} \tag{7.112}$$

Substituting these we get

$$\sigma_s = \frac{2\pi}{k^2} \sum_{l=1}^{\infty} (2l+1) \left(\left| T^{MM}_l \right|^2 + \left| T^{NN}_l \right|^2 \right) \tag{7.113}$$

A factor of two comes from summing the $m = \pm 1$ terms.

7.8.2 Scattering Cross Section of a PEC Sphere

The scattering cross section of a PEC sphere is given by substituting the T-matrix for a PEC sphere into (7.113)

$$\sigma_{s,PEC} = \frac{2\pi}{k^2} \sum_{l=1}^{\infty} (2l+1) \left(\left| \frac{j_l(ka)}{h^{(1)}_l(ka)} \right|^2 + \left| \frac{[kaj_l(ka)]'}{[kah^{(1)}_l(ka)]'} \right|^2 \right) \tag{7.114}$$

where the sphere has radius a in a background with wavenumber k.

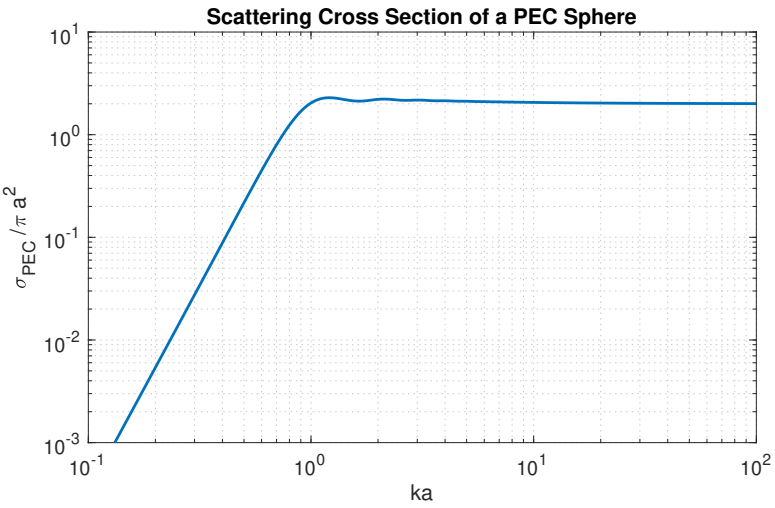


Figure 7.5: Scattering cross section of a PEC sphere, normalized by cross-section area, in free-space.

The routine `scs_pec_sphere` is a workable routine to compute (7.114). The sum is truncated when incremental change is less than machine precision.

```
function [scs] = scs_pec_sphere(k,a)
% Scattering Cross Section of a PEC Sphere
%
% k:        background wavenumber
% a:        sphere radius
%
% scs:  scattering cross section of the PEC sphere
%
% Dependencies: sbesselj, sbesselh, sbesseljp2, sbesselhp2

x = k*a;
l = 1;
err = 1;
old = 0;
scs = 0;
while err > eps;
    j1 = sbesselj(l,x);
    j1p = sbesseljp2(l,x);
    h1 = sbesselh(l,x);
    h1p = sbesselhp2(l,x);
    t1 = abs(j1./h1).^2;
    t2 = abs(j1p./h1p).^2;
    tmp = (2*l+1).*(t1 + t2);
    scs = old + tmp;
    err = abs(tmp)^2/abs(scs)^2;
    old = scs;
    l = l + 1;
end
scs = 2*pi/(abs(k)^2)*scs;
```

7.8.3 Scattering Cross Section of a Dielectric Sphere

The scattering cross section of a dielectric sphere is given by substituting the T-matrix for a dielectric sphere into (7.113). This gives identical results to [10] Figure 1.6.3 and also matches outputs from Ulaby and Long 2014 Code Module 8.12.

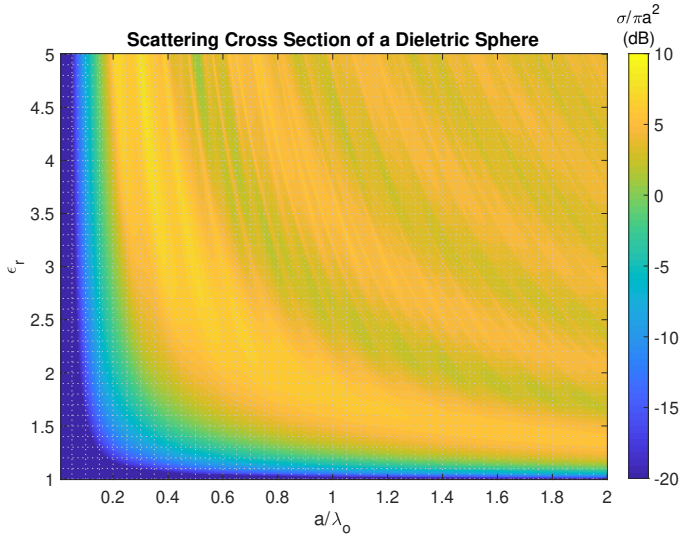


Figure 7.6: Scattering cross section of lossless dielectric sphere, normalized by cross-section area, in free-space.

The routine `scs_dielectric_sphere` computes the scattering cross section of a dielectric sphere given the inner and outer wavenumbers k_1, k_2, and sphere radius a.

```
function [scs] = scs_dieletric_sphere(k1,k2,a)
% Scattering Cross Section of a Dieleitric Sphere
%
% k1:   background wavenumber
% k2:   sphere wavenumber
% a:    sphere radius
%
% scs:  scattering cross section of a dieletric sphere
%
% Dependencies: sbesselj, sbesselh, sbesseljp2, sbesselhp2

x1 = k1*a;
x2 = k2*a;
l = 1;
err = 1;
old = 0;
scs = 0;
while err > eps;
    j1 = sbesselj(l,x1);
    j2 = sbesselj(l,x2);
    j1p = sbesseljp2(l,x1);
    j2p = sbesseljp2(l,x2);
    h1 = sbesselh(l,x1);
    h1p = sbesselhp2(l,x1);
    num1 = j2.*j1p - j1.*j2p;
    den1 = h1.*j2p - j2.*h1p;
    num2 = x2.^2.*j2.*j1p - x1.^2.*j1.*j2p;
    den2 = x1.^2.*h1.*j2p - x2.^2.*j2.*h1p;
    t1 = abs(num1./den1).^2;
    t2 = abs(num2./den2).^2;
    tmp = (2*l+1).*(t1 + t2);
    scs = old + tmp;
    err = abs(tmp)^2/abs(scs)^2;
    old = scs;
    l = l + 1;
end
scs = 2*pi/(abs(k1)^2)*scs;
```

7.8.4 Polarization and Orientation Averaged Scattering Cross Section using T-matrix

We can compute the polarization and orientation averaged scattering cross section directly from the elements of the T-matrix using (6.23) and (7.108):

$$\langle\sigma\rangle = \frac{1}{2\pi}\int_0^{2\pi}\left(\frac{1}{4\pi}\int\sigma_{s\beta}(\hat{k}_i)d\Omega_i\right)d\beta \tag{7.115}$$

Let the incident polarization vector be $\hat{\beta} = \cos\beta\hat{\theta} + \sin\beta\hat{\phi}$, then, for example, the first magnitude squared term in (7.108) looks like

$$\begin{aligned} I_{lm}(\theta_i,\phi_i,\beta) = &\left|\sum_{l'm'} T^{MM}_{lm,l'm'} i^{l'}\left(C^*_{\theta,l'm'}(\theta_i,\phi_i)\cos\beta + C^*_{\phi,l'm'}(\theta_i,\phi_i)\sin\beta\right)\right. \\ &\left. + T^{MN}_{lm,l'm'} i^{l'+1}\left(B^*_{\theta,l'm'}(\theta_i,\phi_i)\cos\beta + B^*_{\phi,l'm'}(\theta_i,\phi_i)\sin\beta\right)\right|^2 \end{aligned} \tag{7.116}$$

After expanding the magnitude squared with a second sum, it is clear that cross terms having $\cos\beta\sin\beta$ will not survive the β integral and can be ignored. From orthogonality of the vector spherical harmonics the cross terms of the vector spherical harmonics will not survive the integration over incident directions. The only terms that matter are of the form $C^*_{\theta,l'm'}(\theta_i,\phi_i)C_{\theta,l''m''}(\theta_i,\phi_i)\cos^2\beta + C^*_{\phi,l'm'}(\theta_i,\phi_i)C_{\phi,l''m''}(\theta_i,\phi_i)\sin^2\beta$. The trick is to first compute the β integral, so that $\cos^2\beta$ and $\sin^2\beta$ become π which is factored out, after which $1/2$ remains from the β integral normalization. The dot products of the vector spherical harmonics then integrate to 1 and collapse one sum. Only the $1/4\pi$ remains from the normalization of the integral over incident directions. The end result is a double sum over the magnitude squared of all elements of the T-matrix:

$$\langle\sigma\rangle = \frac{2\pi}{k^2}\sum_{lm}\sum_{l'm'}\left|T^{MM}_{lm,l'm'}\right|^2 + \left|T^{MN}_{lm,l'm'}\right|^2 + \left|T^{NM}_{lm,l'm'}\right|^2 + \left|T^{NN}_{lm,l'm'}\right|^2 \tag{7.117}$$

Equation (7.117) is basically the square of the Frobenius norm of the entire T-matrix, scaled by a constant that depends on wavelength. This expression is amazingly elegant and comes from the fact that each T-matrix element contains scattering information for all spherical directions and polarizations at once. This can be checked against the scattering cross section of the sphere, which is independent of incident polarization and direction, and therefore $\langle\sigma\rangle = \sigma_s$: the T-matrix elements of the sphere are diagonal (eliminates one sum) with no cross terms and depends only on l, so that each sum over m contributes $2l+1$ terms, which gives (7.113).

The routine `compute_avescs_from_tmatrix` computes the polarization and orientation average scattering cross section from a T-matrix. It takes as input the four $N \times N$ block T-matrices where $N = L^2 + 2L$, for harmonics up to maximum degree L all $\pm m$ and returns (7.117). This is easy enough to compute.

```
function [avescs] = compute_avescs_from_tmatrix(Tmm,Tmn,Tnm,Tnn,k)
% Compute polarization and orientation averaged scattering cross section from T-matrix
%
% Tmm,Tmn,Tnm,Tnn:    T-matrix block matrices
%                     Size [NxN], N = L^2 + 2L
% k:                  Background wavenumber
%
% avescs:             polarization and orientation averaged scattering cross section

avescs = (2*pi/(abs(k)^2))*sum(abs(Tmm(:)).^2 + abs(Tmn(:)).^2 ...
    + abs(Tnm(:)).^2 + abs(Tnn(:)).^2);
```

Chapter 8

Fast Multipole Method

The fast multipole method (FMM) was developed to accelerate the matrix-vector multiplication of the impedance matrix in an iterative Method of Moments scattering solution. The FMM is built on two ideas: 1) the band-limited nature of far-field patterns, 2) the plane wave expansion of the Green's function. Together, these allow the scattered fields of any number of individual scatterers to be collected into a single scattering pattern and translated to a different reference frame at constant computational cost. It was originally developed by Greengard and Rokhlin [34]. Following this, the multi-level fast multipole method (MLFMM), developed by Song and Chew [35], uses a hierarchical structure to aggregate, translate, and disaggregate fields up and down a tree structure between small and large length scales to accomplish the computation in $O(N \log N)$ time. The FMM is basically the FFT of scattering.

The title of this chapter is a little misleading, because we will not actually implement the MLFMM and MoM. We will however give routines for many of the core operations that are incredibly useful for manipulating scalar and vector spherical harmonics and aggregating/disaggregating scattered field patterns. In fact, the original motivation for coding these routines was to aggregate/disaggregate far-field scattering patterns in multi-layer radar facet models.

The routines in this chapter are based largely on the derivations in [36], which is one of the best and clearest references for the FMM and its constituent parts. We add modifications, alternative explanations, and corrections where needed, and skip much of the formality in order to get to the details of the computations. Accurately computing the FMM in full is quite complicated, and explanations are better left to the FMM literature.

An overview of the FMM formulation is first given. We explain quadrature integration over the sphere, on which most of the computations are based. We then discuss concepts of field aggregation and disaggregation. We give codes for the translation operator and interpolation. A number of routines are provided for interpolating and filtering scalar and vector spherical field patterns. We then give routines for S-matrix and T-matrix transforms based on the FMM spherical harmonic filters. Finally, a routine for computing a version of the 1D FMM is given, which can be used to accelerate the spherical filters.

8.1 Far-Field Green's Function and Plane Wave Expansion

The core for the fast multipole method lies in the plane wave expansion of the kernel of the scalar Green's function. This is derived and explained in excellent detail in [36]. We summarized the main points here.

Far-Field Green's Function Recall that the electric field dyadic Green's function is given by

$$\overline{\mathbf{G}}(\mathbf{r}, \mathbf{r}') = \left[\overline{\mathbf{I}} + \frac{1}{k^2}\nabla\nabla\right] g(\mathbf{r}, \mathbf{r}') \tag{8.1}$$

where the scalar Green's function is

$$g(\mathbf{r}, \mathbf{r}') = \frac{e^{ik|\mathbf{r}-\mathbf{r}'|}}{4\pi|\mathbf{r}-\mathbf{r}'|} \tag{8.2}$$

and that the far-field dyadic Green's function is

$$\overline{\mathbf{G}}_f(\mathbf{r},\mathbf{r}') \approx \left[\bar{\mathbf{I}} - \hat{\mathbf{k}}\hat{\mathbf{k}}\right] g(\mathbf{r},\mathbf{r}') \tag{8.3}$$

where in using (8.2) we have not approximated the phase term.

Plane Wave Expansion The derivation is based on two expansions. The first an expansion of the kernel of the scalar Green's function, [36],

$$\frac{e^{ik|\mathbf{X}+\mathbf{d}|}}{|\mathbf{X}+\mathbf{d}|} = ikh_0^{(1)}\left(k\,|\mathbf{X}+\mathbf{d}|\right) \tag{8.4}$$

$$= ik\sum_{l=0}^{\infty}(-1)^l(2l+1)j_l(kd)h_l^{(1)}(kX)P_l\left(\hat{\mathbf{d}}\cdot\hat{\mathbf{X}}\right) \tag{8.5}$$

where k is the free-space wavenumber, j_l is the spherical Bessel function, $h_l^{(1)}$ is the spherical Hankel function, and P_l is the Legendre polynomial. The expansion is valid for $d < X$ where $d = |\mathbf{d}|$ and $X = |\mathbf{X}|$. The second expansion is

$$j_l(kd)P_l\left(\hat{\mathbf{d}}\cdot\hat{\mathbf{X}}\right) = \frac{i^{-l}}{4\pi}\int e^{i\mathbf{k}\cdot\mathbf{d}}P_l(\hat{\mathbf{k}}\cdot\hat{\mathbf{X}})d\Omega_k \tag{8.6}$$

where the integral is over the sphere of plane wave directions, $\hat{\mathbf{k}} = \mathbf{k}/k = (\sin\theta_k\cos\phi_k, \sin\theta_k\sin\phi_k, \cos\theta_k)$ and differential $d\Omega_k = d^2\hat{\mathbf{k}} = \sin\theta_k d\theta_k d\phi_k$. Substituting (8.6) into (8.5)

$$\frac{e^{ik|\mathbf{X}+\mathbf{d}|}}{|\mathbf{X}+\mathbf{d}|} = \frac{ik}{4\pi}\int e^{i\mathbf{k}\cdot\mathbf{d}}\sum_{l=0}^{\infty} i^l(2l+1)h_l^{(1)}(kX)P_l(\hat{\mathbf{k}}\cdot\hat{\mathbf{X}})d\Omega_k \tag{8.7}$$

Next, let source and observation points be $\mathbf{r}'$ and $\mathbf{r}$ with associated local centers, $\mathbf{r}_s$ and $\mathbf{r}_o$, respectively. With these, the vectors $\mathbf{X}$ and $\mathbf{d}$ are defined

$$\mathbf{X} = \mathbf{r}_o - \mathbf{r}_s \tag{8.8}$$

$$\mathbf{d} = \mathbf{r} - \mathbf{r}_o - (\mathbf{r}' - \mathbf{r}_s) \tag{8.9}$$

The vector $\mathbf{X}$ points from the local center of the source points to the local center of the observation points. The vector $\mathbf{d}$ is a vector that would point from the source point to the observation point if the two local regions were translated so that they overlapped. Under the validity condition for the sum, the two local regions must be non-overlapping spheres, in other words $|\mathbf{r} - \mathbf{r}_o| < X/2$ and $|\mathbf{r}' - \mathbf{r}_s| < X/2$.

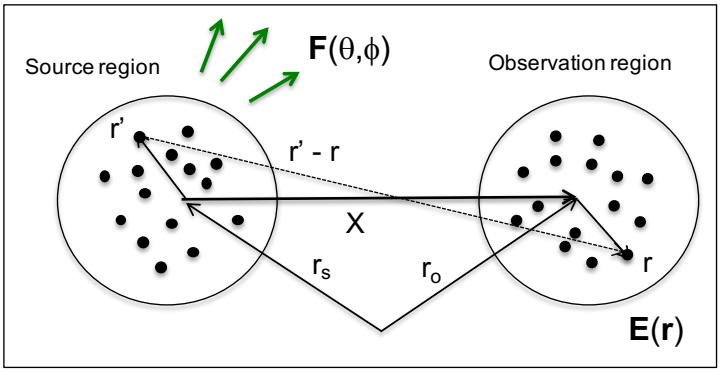


Figure 8.1: Geometry for the plane wave expansion of the scalar and dyadic Green's functions.

Substituting (8.8) and (8.9) into (8.7), and after truncating the sum at degree L, the kernel can be approximated

$$\frac{e^{ik|\mathbf{r}-\mathbf{r}'|}}{|\mathbf{r}-\mathbf{r}'|} \approx \frac{ik}{4\pi}\int e^{i\mathbf{k}\cdot(\mathbf{r}-\mathbf{r}_o)}T_L(\mathbf{k},\mathbf{X})e^{-i\mathbf{k}\cdot(\mathbf{r}'-\mathbf{r}_s)}d\Omega_k \tag{8.10}$$

where T_L is the translation operator given by

$$T_L(\mathbf{k},\mathbf{X}) = \sum_{l=0}^{L} i^l(2l+1)h_l^{(1)}(kX)P_l(\hat{\mathbf{k}}\cdot\hat{\mathbf{X}}) \tag{8.11}$$

Green's Functions Using (8.10), the scalar Green's function can be written

$$g(\mathbf{r},\mathbf{r}') \approx \frac{ik}{16\pi^2}\int e^{i\mathbf{k}\cdot(\mathbf{r}-\mathbf{r}_o)}T_L(\mathbf{k},\mathbf{X})e^{-i\mathbf{k}\cdot(\mathbf{r}'-\mathbf{r}_s)}d\Omega_k \tag{8.12}$$

From which the far-field dyadic Green's function is approximated

$$\overline{\mathbf{G}}_f(\mathbf{r},\mathbf{r}') \approx \frac{ik}{16\pi^2}\int \left[\bar{\mathbf{I}}-\hat{\mathbf{k}}\hat{\mathbf{k}}\right] e^{i\mathbf{k}\cdot(\mathbf{r}-\mathbf{r}_o)}T_L(\mathbf{k},\mathbf{X})e^{-i\mathbf{k}\cdot(\mathbf{r}'-\mathbf{r}_s)}d\Omega_k \tag{8.13}$$

Treating $\hat{\mathbf{k}}$ as the radial unit vector, the vector dyad can be written $\bar{\mathbf{I}}-\hat{\mathbf{k}}\hat{\mathbf{k}} = \hat{\boldsymbol{\theta}}\hat{\boldsymbol{\theta}}+\hat{\boldsymbol{\phi}}\hat{\boldsymbol{\phi}}$, which shows that the far pattern has only $\hat{\boldsymbol{\theta}}$ and $\hat{\boldsymbol{\phi}}$ vector components. Note, the polarization vectors in the dyadic Green's function are relative to the local center of the source and are not integrated because the integral only expands the scalar part of the kernel.

To illustrate how the dyadic Green's function expansion is used with a source, consider the electric field given by the volume integral

$$\mathbf{E}(\mathbf{r}) = i\omega\mu\int \overline{\mathbf{G}}(\mathbf{r},\mathbf{r}')\cdot\mathbf{J}(\mathbf{r}')dV \tag{8.14}$$

where $\mathbf{J}$ is the current density. Substituting (8.13), we can write this as

$$\mathbf{E}(\mathbf{r}) \approx \frac{ik}{4\pi}\int \mathbf{F}(\hat{\mathbf{k}})e^{i\mathbf{k}\cdot(\mathbf{r}-\mathbf{r}_o)}T_L(\mathbf{k},\mathbf{X})d\Omega_k \tag{8.15}$$

where $\mathbf{F}(\hat{\mathbf{k}})$ is the far-field radiation pattern of the source

$$\mathbf{F}(\hat{\mathbf{k}}) = \frac{1}{4\pi}(i\omega\mu)\left[\bar{\mathbf{I}}-\hat{\mathbf{k}}\hat{\mathbf{k}}\right]\cdot\int e^{-i\mathbf{k}\cdot(\mathbf{r}'-\mathbf{r}_s)}\mathbf{J}(\mathbf{r}')dV \tag{8.16}$$

A similar form can be found in [37]. In other words, given a far-field vector radiation pattern, which could equally be that of a scatterer, the electric field at observation point $\mathbf{r}$ is computed by (8.15), which integrates the product of the pattern, plane wave phases, and translation matrix over all plane wave directions.

8.2 Selection of L

In general, the maximum degree L that is required to accurately compute the translation operation is proportional to the dimension of the source/observation spheres. There are various formulas for L. From [38, 36], one formula is

$$L \approx kd + \beta\ln(\pi+kd) \tag{8.17}$$

where β is the number of digits of precision. From [39, 36], the excess bandwidth formula is

$$L \approx kd + 1.8\alpha^{2/3}(kd)^{1/3} \tag{8.18}$$

where $\alpha = \log_{10}(1/\epsilon)$, and ϵ is the number of digits of precision. In both case, L should be rounded up.

It needs to be noted that the sum of the translation operator does not become more accurate with more harmonics. In fact, it will break down if the number of harmonics is excessively large. This is due to unstable summation of the Hankel functions when L is too large relative to the argument. Therefore, there is a balance between enough harmonics for accurate translation and too many of them that render the sum inaccurate. This is explained in detail in [36].

8.3 Integration over the Unit Sphere

Here we explain rules for sampling and integrating spherical harmonics over the sphere. This is needed for understanding how to compute the plane wave expansions of the FMM, as well as spherical harmonic interpolation and filtering. This is based on a hybrid of Gauss-Legendre quadrature integration in θ and trapezoidal integration in ϕ. It is exact for band-limited spherical functions assuming a minimum number of sampling points is used, which we derive next. While the method is exact and relatively simple, more efficient spherical integration schemes do exist and that use fewer integration points.

8.3.1 Gauss-Legendre quadrature

In general, quadrature is used to compute an integral of a continuous function as a weighted sum of its samples. Gauss-Legendre quadrature (Gaussian quadrature) is exact for polynomials of degree $2n-1$ with n nodes and weights, x_j and w_j, respectively. This is normally presented on the domain $x=[-1,1]$ as

$$\int_{-1}^{1} f(x)dx = \sum_{j}^{n} w_j f(x_j) \tag{8.19}$$

The nodes are given by the jth zero of the Legendre polynomials $P_n(x_j)$ normalized such that $P_n(1)=1$, with weights

$$w_j = \frac{2}{(1-x_j^2)\left[P_n'(x_j)\right]^2} \tag{8.20}$$

Routines exist for computing the nodes and weights of Gauss-Legendre quadrature. We recommend the routine `legpts` from the `http://www.chebfun.org/` library, which we use throughout and do not repeat here.

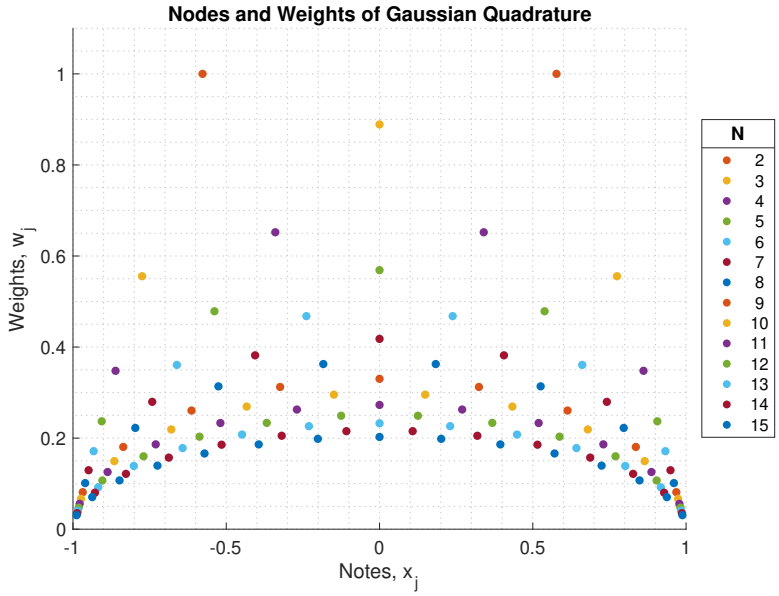


Figure 8.2: Notes and weights of Gaussian quadrature

8.3.2 Numerical Integration of Spherical Harmonics

Here we derive the number of sampling points needed to numerically integrate spherical harmonics, which follows the results in [40]. Let $f(\theta,\phi)$ be a spherical scalar function composed of a finite number of spherical

harmonics

$$f(\theta,\phi) = \sum_{l=0}^{L}\sum_{m=-l}^{l} f_{lm} Y_{lm}(\theta,\phi) \tag{8.21}$$

Integrating this over the unit sphere and separating variables

$$\int_0^{2\pi}\int_0^{\pi} f(\theta,\phi)\sin\theta d\theta d\phi = \frac{1}{\sqrt{2\pi}}\sum_{l=0}^{L}\sum_{m=-l}^{l} f_{lm}\int_0^{2\pi} e^{im\phi}d\phi\int_0^{\pi}\widetilde{P}_l^m(\cos\theta)\sin\theta d\theta d\phi \tag{8.22}$$

$$\tag{8.23}$$

The integral over ϕ can be computed analytically as

$$\int_0^{2\pi} e^{im\phi}d\phi = \begin{cases} 2\pi, & m=0 \\ \dfrac{i(1-e^{2i\pi m})}{m} = 0, & m\neq 0 \end{cases} \tag{8.24}$$

It is clear that when $m \neq 0$, the double integral will be zero regardless of the value of the θ integral. We still want to know the number of discrete integration points in ϕ required to make this true. Using trapezoidal integration with periodicity, (8.24) is written as a discrete sum over N evenly spaced points

$$\int_0^{2\pi} e^{im\phi}d\phi = \Delta\phi\sum_{k=0}^{N-1} e^{im\phi_k} \tag{8.25}$$

$$= \frac{2\pi}{N}\sum_{k=1}^{N} e^{imk2\pi/N} \tag{8.26}$$

$$= \frac{2\pi}{N}e^{i(N-1)m\pi/N}\frac{\sin(m\pi)}{\sin(m\pi/N)} \tag{8.27}$$

where $\phi_k = (k-1)\Delta\phi$, $k = 1, ..., N$, and $\Delta\phi = 2\pi/N$, The last equation comes from the Dirichlet kernel

$$\sum_{k=0}^{N-1} e^{ikx} = e^{i(N-1)x/2}\frac{\sin(Nx/2)}{\sin(x/2)} \tag{8.28}$$

(8.27) will be zero when $|m| < N$, because the numerator sine is zero. When $m = N$, applying L'Hopital's rule, the ratio of sine functions is equal to N while the complex exponent is non-zero. Therefore, the number of samples that correctly integrates all harmonics up to $m = L$ is $N = L + 1$. This can be confirmed numerically and is the same as given in [40, 41]. We can then write the ϕ integral as

$$\int_0^{2\pi} e^{im\phi}d\phi = \frac{2\pi}{L+1}\sum_{i=1}^{L+1} e^{im\phi_i}, \quad 0 \leq |m| \leq L \tag{8.29}$$

When $m = 0$, the θ integral becomes

$$\int_0^{\pi} P_l(\cos\theta)\sin\theta d\theta = -\int_0^{\pi} P_l(\cos\theta)d\cos\theta \tag{8.30}$$

$$= \int_{-1}^{1} P_l(\mu)d\mu \tag{8.31}$$

$$= \sum_{j=1}^{N} w_j P_l(\mu_j) \tag{8.32}$$

with the change of variables $\mu = \cos\theta$ and where the integral has been replaced with Gaussian quadrature. Because $P_l(\mu)$ are polynomials of degree l, and because the quadrature is exact for polynomial degrees less

than $2n-1$, the number of points that will correctly integrate this is $l < 2n-1$. For maximum harmonic degree L, the number of quadrature nodes is $N = (L+1)/2$, which should be rounded up.

As an aside, when m is even, the associated Legendre polynomials can be integrated with quadrature, because they are simple polynomials. When m is odd, they contain a factor of $\sqrt{1-\mu^2}$, which means they are not simple polynomials, and so cannot be integrated exactly via quadrature. This can be verified numerically. However, analytical integration of $P_l^m(\mu)$ can be done for any m, [41, 42].

Using these results, numerical integration of a spherical function composed of spherical harmonics with maximum degree L can be computed exactly (to machine precision) as

$$\int_0^{2\pi}\int_0^{\pi} f(\theta,\phi)\sin\theta d\theta d\phi = \frac{2\pi}{L+1}\sum_{i=1}^{L+1}\sum_{j=1}^{\lceil (L+1)/2\rceil} w_j f(\theta_j,\phi_i) \tag{8.33}$$

where $\phi_i = (i-1)2\pi/(L+1)$, $i = 1, ..., L+1$, and $\theta_j = \arccos\mu_j$, where μ_j and w_j are the nodes and weights of Gaussian quadrature for $\lceil (L+1)/2\rceil$ points.

We know that the spherical harmonics are zero-mean over the sphere except the monopole, therefore, if the expansion coefficients are known, one can simply use f_{00} for the mean. If the coefficients are not known, but the function is sampled on the points of quadrature, (8.33) will compute the mean of $f(\theta,\phi)$ exactly.

8.3.3 Numerical Integration of Products of Spherical Harmonics

Here we derive the number of sampling points needed to numerically integrate a product of spherical harmonics over the unit sphere. This can be done using spherical harmonic synthesis. The expansion coefficients of a scalar spherical function are given by

$$f_{lm} = \int_0^{2\pi}\int_0^{\pi} f(\theta,\phi)Y_{lm}^*(\theta,\phi)\sin\theta d\theta d\phi \tag{8.34}$$

Substituting (8.21) (ignore for the moment equivalency and orthogonality)

$$f_{lm} = \sum_{l'=0}^{L}\sum_{m'=-l'}^{l'} f_{l'm'}\frac{1}{2\pi}\int_0^{2\pi} e^{i(m'-m)\phi}d\phi\int_0^{\pi}\widetilde{P}_{l'}^{m'}(\cos\theta)\widetilde{P}_l^m(\cos\theta)\sin\theta d\theta \tag{8.35}$$

$$\tag{8.36}$$

Using the reasoning in the previous section, the maximum harmonic in the ϕ integration is $2L$. Therefore the number of equally spaced sampling points that are required for trapezoidal integration in ϕ is $2L+1$. The product of two associated Legendre polynomials is a pure polynomial of degree $2L$ for any m. Therefore, the number of required samples for Gaussian quadrature in θ is $\lceil L+1/2\rceil$, which can be immediately rounded up to $L+1$.

Using these, numerical integration of the product of two spherical functions, $f(\theta,\phi)$ and $g(\theta,\phi)$, each with maximum harmonic degree L can be computed exactly as

$$\int_0^{2\pi}\int_0^{\pi} f(\theta,\phi)g(\theta,\phi)\sin\theta d\theta d\phi = \frac{2\pi}{2L+1}\sum_{i=1}^{2L+1}\sum_{j=1}^{L+1} w_j f(\theta_j,\phi_i)g(\theta_j,\phi_i) \tag{8.37}$$

where $\phi_i = (i-1)2\pi/(2L+1)$, $i = 1, ..., 2L+1$, and $\theta_j = \arccos\mu_j$, where μ_j and w_j are the nodes and weights of Gaussian quadrature for $L+1$ points. It is common to find (8.37) in the literature applied to spherical functions without stipulating whether the underlying function is composed of pure harmonics of a product of spherical harmonics.

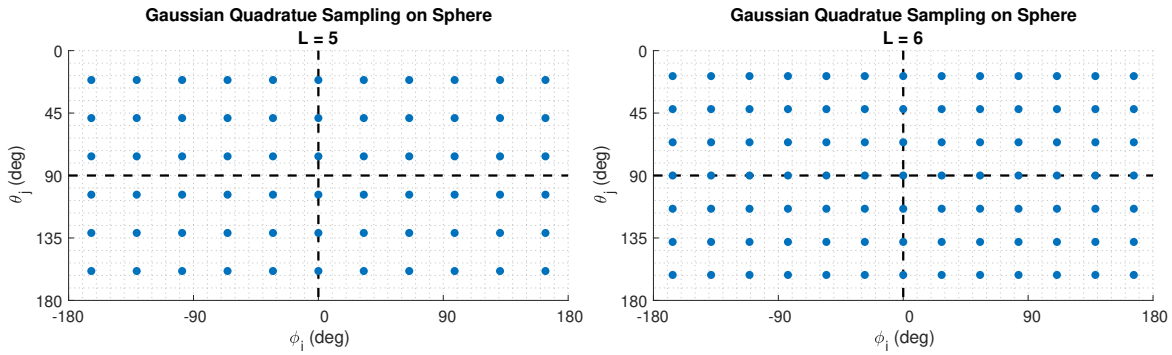


Figure 8.3: Nodes of trapezoidal integration (ϕ_i) and Gaussian quadrature (θ_j) for integrating products of spherical functions.

The nodes θ_j are almost uniformly spaced, and they never sample the poles because of the nodes of Gaussian quadrature do not sample the end points. This is especially convenient for vector spherical harmonics where the polarization is ambiguous at the poles. When $L+1$ is odd, the nodes will sample the equator. This scheme does crowd the poles somewhat.

Finally, (8.37) can be viewed as computing the power of a field over the sphere when the second function is conjugated (or computing the cross-correlation of two fields). If the spherical harmonic expansion coefficients are known, the analogous form of Parseval's theorem can be used to simply sum the magnitude squared of the coefficients. If the coefficients are not known, (8.37) will compute a power-like quantity exactly from samples of the field(s).

8.3.4 Green's Function Integration

Following [36], we give the number of sample points required to correctly integrate the plane wave expansions in the FMM. Using the addition theorem for plane waves

$$e^{i\mathbf{k}\cdot\mathbf{d}} = \sum_{l=0}^{\infty} i^l (2l+1) j_l(kd) P_l(\hat{\mathbf{k}}\cdot\hat{\mathbf{d}}) \tag{8.38}$$

(8.7) can be written

$$\frac{e^{ik|\mathbf{X}+\mathbf{d}|}}{|\mathbf{X}+\mathbf{d}|} = \frac{ik}{4\pi} \sum_{l=0}^{\infty}\sum_{l'=0}^{\infty} i^l (2l+1) i^{l'} (2l'+1) h_l^{(1)}(kX) j_l'(kd) \int P_l(\hat{\mathbf{k}}\cdot\hat{\mathbf{X}}) P_l'(\hat{\mathbf{k}}\cdot\hat{\mathbf{d}}) d\Omega_k \tag{8.39}$$

Using the addition theorem for Legendre polynomials,

$$P_l(\hat{\mathbf{r}}\cdot\hat{\mathbf{r}}') = \frac{4\pi}{2l+1} \sum_{m=-l}^{l} Y_{lm}(\theta,\phi) Y_{lm}^*(\theta',\phi') \tag{8.40}$$

the integral can be expanded as

$$\int P_l(\hat{\mathbf{k}}\cdot\hat{\mathbf{X}}) P_l'(\hat{\mathbf{k}}\cdot\hat{\mathbf{d}}) d\Omega_k = \frac{4\pi}{2l+1}\frac{4\pi}{2l'+1} \sum_{m=-l}^{l}\sum_{m=-l'}^{l'} Y_{lm}^*(\theta_X',\phi_X') Y_{lm}^*(\theta_d',\phi_d') \int \left(Y_{lm}(\theta_k,\phi_k)\right)^2 d\Omega_k \tag{8.41}$$

Which shows that the spherical integral in (8.7) is really integrating a product of spherical harmonics. Therefore, using the results from the previous section, the integral over planes waves in the Green's function kernel (8.10) can be computed exactly over discrete values of the wave vector $\hat{\mathbf{k}}_{ij}$ as

$$\frac{e^{ik|\mathbf{r}-\mathbf{r}'|}}{|\mathbf{r}-\mathbf{r}'|} \approx \frac{ik}{4\pi}\frac{2\pi}{2L+1} \sum_{i=1}^{2L+1}\sum_{j=1}^{L+1} w_j e^{ik\hat{\mathbf{k}}_{ij}\cdot(\mathbf{r}-\mathbf{r}_o)} T_L(k\hat{\mathbf{k}}_{ij},\mathbf{X}) e^{-ik\hat{\mathbf{k}}_{ij}\cdot(\mathbf{r}'-\mathbf{r}_s)} \tag{8.42}$$

which is the result in [36]. The approximation comes from truncating the sum in (8.10), not the spherical integration.

8.4 Aggregation/Disaggregation

Here we give a basic idea of how to aggregate, translate, and disaggregate fields in the context of the FMM operations following the explanation in [36]. The routines for computing the translation operator are given in Section 8.5, while routines for interpolating and filtering scalar and vector fields are given in Sections 8.6 and 8.7.

8.4.1 Octree

Typically the scatterers in an FMM problem are organized on a hierarchical octree. An octree is a data structure in which each node has eight children. This is combined with the geometric process of subdividing a cubic volume into eight equal octants. The eight subcubes are called the children of the larger parent cube and visa versa. A cube at every level has an outgoing and incoming far-field pattern associated with it, say $\mathbf{F}(\hat{\mathbf{k}})$, which is sampled on the sphere according to the rules of quadrature. This field expansion is centered on the cube and has harmonic bandwidth (i.e., maximum degree vector spherical harmonic, L) at least as large as that required for the diameter of the enclosing sphere, and further set by the desired accuracy of the translation operations. This means that fields are coarsely sampled in (θ, ϕ) at higher levels (smaller cubes), and more finely sampled at lower (larger cubes) levels.

Fields are aggregated up the hierarchy, translated at the highest level possible, then disaggregated down the hierarchy. There is a constraint that fields cannot be translated to neighboring boxes at the same level (due to the separation requirement of the translation). This creates a complication when disaggregating. For more details see [36]. In general, aggregation and disaggregation do not have to be restricted to octree structures as long as the bandwidth and separation between the groups of scatterers is obeyed.

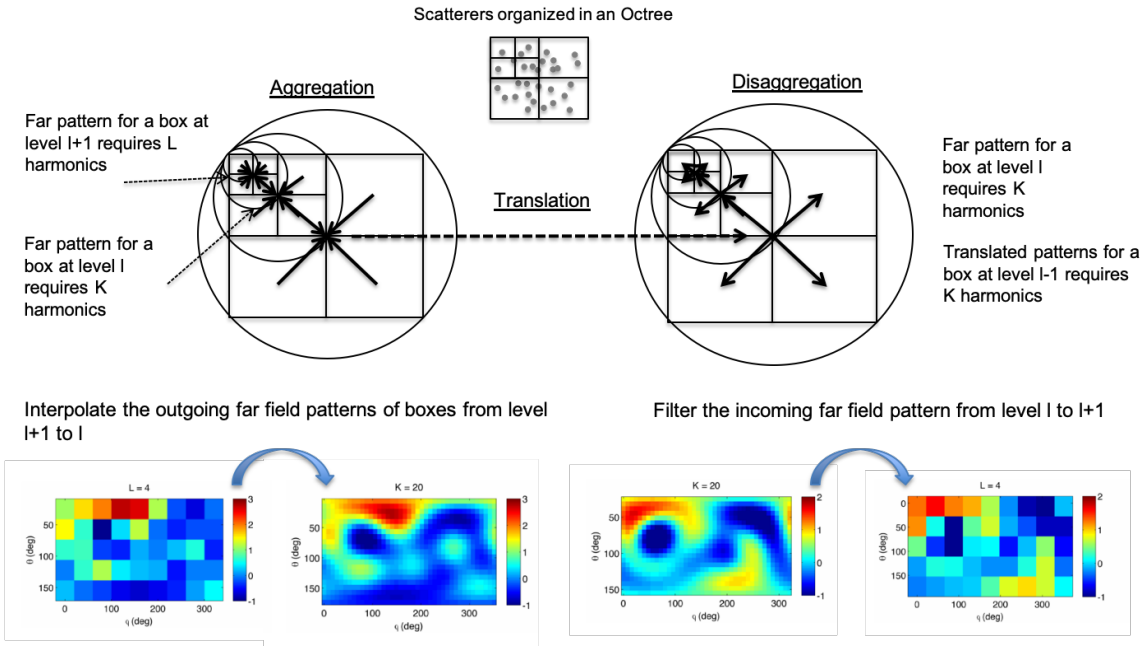


Figure 8.4: Aggregation, translation, and disaggregation.

8.4.2 Aggregation

To reiterate, the far pattern of each child cube has lower harmonic content than the parent (due to its smaller size), and therefore coarser spatial sampling in (θ, ϕ). The process of aggregating fields consists of 1) interpolating each far pattern of the children cubes up to the finer sampling of the parent, 2) shifting the phase centers of the children's patterns to that of the parent, then 3) summing the fields of the all

the children. The shift is done by multiplication by a complex exponential of plane wave phases, which is equivalent to a diagonal matrix-vector multiply and is trivial to compute.

Let $\mathbf{F}_{n,l}(\hat{\mathbf{k}})$ be the vector field for the nth group (cube) at level l. Let P_{l+1}^{l} be the interpolation operator that interpolates a field from level $l+1$ to level l. The aggregated field for the nth group at the level of the parents is the sum over all interpolated and shifted fields of the children belonging to each parent, [36]:

$$\mathbf{F}_{n,l}(\hat{\mathbf{k}}_l) = \sum_{m \in G_c} e^{i\mathbf{k}_{l+1}\cdot(\mathbf{x}_n - \mathbf{x}_m)} P_{l+1}^{l} \left[\mathbf{F}_{m,l+1}(\hat{\mathbf{k}}_{l+1}) \right] \tag{8.43}$$

where G_c are the list of children that belong to parent group n, and $\hat{\mathbf{k}}_l$ are the spherical directions sampled for level l.

The process of interpolation does not change the harmonic content of the patterns of the children. However, multiplying the pattern by the phase exponential of the plane wave shift is equivalent to convolving the spherical harmonic spectra. This is why the fields are first interpolated, then translated. Another way to think about this is, even though the patterns of the children may contain lower harmonic content when centered on the cubes of the children, the same pattern that is offset from a different center, now belongs to a larger enclosing sphere, and therefore has more harmonic content requiring finer spherical sampling.

8.4.3 Disaggregation

Disaggregation sweeps from the lowest level of the octree (largest cubes) to the highest level (smallest cubes) and consists of three steps: 1) shift the field of a parent to the phase center of the child then filter the parent's field to the child's level (i.e., anterpolate), 2) translate the outgoing patterns between groups at the same level that are not near-neighbors but whose parents are near-neighbors (i.e., called the neighborhood of the child), 3) sum the filtered and translated fields. This is done recursively from the bottom level to the top level for all groups and can be written.

$$\mathbf{G}_{m,l}(\hat{\mathbf{k}}_l) = P_{l-1}^{l} \left[e^{i\mathbf{k}_{l-1}\cdot(\mathbf{x}_m - \mathbf{x}_n)} \mathbf{G}_{n,l-1}(\hat{\mathbf{k}}_{l-1}) \right] + \sum_{p \in G_w} T_L(\mathbf{k}_l, \mathbf{x}_m - \mathbf{x}_p) \mathbf{F}_{p,l}(\hat{\mathbf{k}}) \tag{8.44}$$

where P_{l-1}^{l} is the filtering operator that filters the parent's field at level $l-1$ to the child sampling, m is index of the child at level l, n is the index of the parent at the parent's level, and G_w is the list children at level l that are well-separated from m (i.e., children that are not near-neighbors, but whose parents are near-neighbors). The purpose of filtering the field of the parent is to reduce its total harmonic content to be of the same degree as that of the child.

8.5 Translation Operator

8.5.1 Basic Translation Operator

The FMM translation operator is

$$T_L(\mathbf{k}, \mathbf{X}) = \sum_{l=0}^{L} i^l (2l+1) h_l^{(1)}(kX) P_l(\hat{\mathbf{k}} \cdot \hat{\mathbf{X}}) \tag{8.45}$$

where $\mathbf{X}$ is the Cartesian vector that points from the origin of the source frame to the origin of the observation frame, k is the complex background wavenumber, $\hat{\mathbf{k}}$ is the Cartesian wave vector direction, $P_l(x)$ is the Legendre polynomial, and L is the maximum degree of the sum. When computing this, it can be written terms of the dot product $\cos\theta = \hat{\mathbf{k}} \cdot \hat{\mathbf{X}}$ in order to externalize the vector computations.

$$T_L(kX, \theta) = \sum_{l=0}^{L} i^l (2l+1) h_l^{(1)}(kX) P_l(\cos\theta) \tag{8.46}$$

The routine `TLth` returns the translation operator (8.46) given scalars kX, L, and array of $\cos\theta$, which can be any size. To save memory, the Legendre polynomials are computed inline with the recursion (12.15).

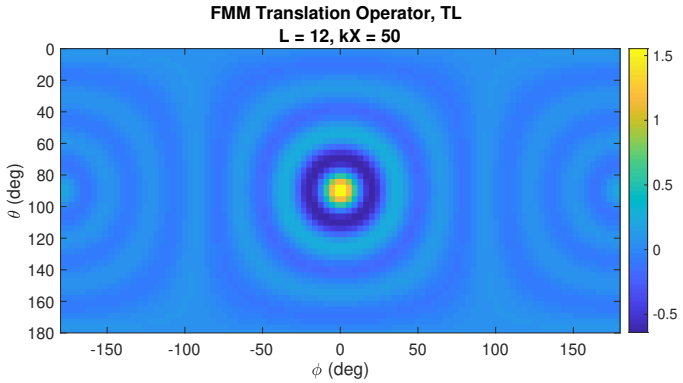


Figure 8.5: Real part of the translation operator, (8.46), for $\hat{\mathbf{X}} = [1, 0, 0]$, $L = 12$, and $kX = 50$. The grid is highly oversampled compared to the sampling required for quadrature integration over the sphere. Note, the operator is peaked in the direction of propagation and contains a 'back lobe'-like feature.

```
function [tl] = TLth(L,kX,costheta)
% FMM Far-field translation operator
%
% T_L = sum_{l=0}^L i^l(2l+1)h_l^{(1)}(kX)P_l(cos(theta))
%
% L:            Maximum order L
% kX:           Product of wavenumber k, and translation magnitude X
% costheta:     Dot product of khat and Xhat
%
% tl:           Translation operator values, size(costheta)
%
% Dependencies: sbesselh

sz = size(costheta);
costheta = costheta(:);
l = (0:L)';
H = sbesselh(l,kX);
H = ((1i).^(l).*(2*l+1)).*H;
tl = H(1)*ones(size(costheta));
if L == 0
    tl = reshape(tl,sz);
    return
end
tl = tl + H(2)*costheta;
if L == 1
    tl = reshape(tl,sz);
    return
end
Pl = 1.5*(costheta.^2) - 0.5;
tl = tl + H(3)*Pl;
if L == 2
    tl = reshape(tl,sz);
    return
end
Plm1 = costheta;
for n=3:L,
    Plm2 = Plm1;
    Plm1 = Pl;
    c1 = (2*n-1)/n;
    c2 = -(n-1)/n;
    Pl = c1*(costheta.*Plm1) + c2*Plm2;
    tl = tl + H(n+1)*Pl;
end
tl = reshape(tl,sz);
```

8.5.2 Translation Operator Interpolation

Computing the translation operator as a straight sum is a computational bottleneck for large problems. Much work has gone into finding an optimal computation scheme, and the result is a fast interpolator. The translation operator is precomputed directly at a coarse sampling, after which any value is found by interpolation to a selectable level of error. Because the translation operator is band limited, it can be computed exactly from the samples using the approximate prolate spheroid (APS) method. In practice, only a small subset of samples in the vicinity of the interpolation point needs to be used.

The interpolation formula using APS is given by [43, 36]

$$\widetilde{T}_L(\theta) = \sum_{m=m_o-p+1}^{m_o+p} T_L(m\Delta\theta) S_N(\theta - m\Delta\theta, \theta_o) D_M(\theta - m\Delta\theta) \tag{8.47}$$

where $\widetilde{T}_L(\theta)$ is the interpolated translation operator, $D_M(\theta)$ is the periodic sinc function (or Dirichlet kernel), $S_N(\theta, \theta_o)$ is a windowing function, and $T_L(m\Delta\theta)$ are precomputed samples of the translation operator. The windowing function is given by

$$S_N(\theta, \theta_o) = \frac{R_N(\theta, \theta_o)}{R_N(0, \theta_o)} \tag{8.48}$$

$$R_N(\theta, \theta_o) = \frac{\sinh\left[(2N+1)\sinh^{-1}\sqrt{\sin^2(\theta_o/2) - \sin^2(\theta/2)}\right]}{\sqrt{\sin^2(\theta_o/2) - \sin^2(\theta/2)}} \tag{8.49}$$

The Dirichlet kernel is given by

$$D_M(\theta) = \frac{\sin\left[(2M+1)\theta/2\right]}{(2M+1)\sin(\theta/2)} \tag{8.50}$$

In these expressions, L is the truncation degree of the sum and $M = sL$ is the total number of precomputed sampling points where s is the over-sampling ratio and is an integer. The required sample spacing is $\Delta\theta = 2\pi/(2M+1)$. This spacing is over a 2π circumference, even though we only need $\theta = [0, \pi]$. This comes from the original papers on optimal interpolation over a sphere, but the formulation persists in the literature. $N = M - L = (s-1)L$ is the number of over-sampling points. $m_o = \text{Int}[\theta/\Delta\theta]$ is the integer index to the left of the interpolation point, where $\text{Int}[\cdot]$ is the integer part or floor function. $\theta_o = p\Delta\theta$ is the width of the interpolation window, where p is the number of samples on each side of the interpolation point. The choice of s and p is important for maintaining accuracy while minimizing computation. Good empirical values are $s = 5$, $p = 3$.

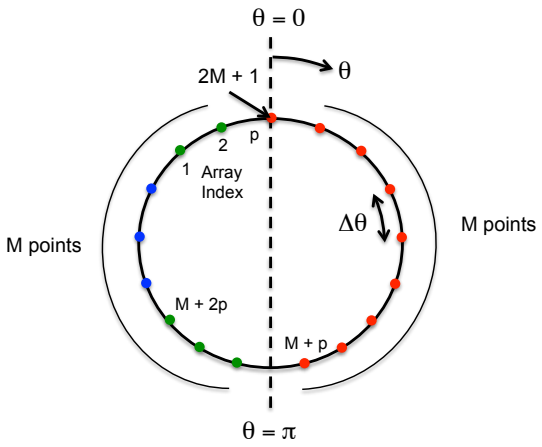


Figure 8.6: Sampling and indexing of the interpolation. Example for $M = 8$, $p = 3$. $m = 0$ corresponds to $\theta = 0$. Note no sample at $\theta = \pi$.

Even though we will only interpolate $\theta = [0, \pi]$, we require precomputed samples outside this range when interpolating near the ends. The translation operator is an even function of θ, therefore, there are two options to obtain the out of bounds points: 1) Only compute sampling points in the range $\theta = [0, \pi]$ and loop the summation index m back on itself if we go beyond the ends, or 2) precompute the necessary values outside of the range, and let the index roam free. We choose the first for simplicity.

Figure 8.6 illustrates the sample spacing as it relates to the number of sample points as well as the indexing scheme for precomputing points outside the range $\theta = [0, \pi]$. There are $p - 1$ samples to the left of 0, p samples after π, $M + 2p$ total sample points, and the array index is $I = m + p$, where $m = [0, M]$. Figure 8.7 shows the interpolator.

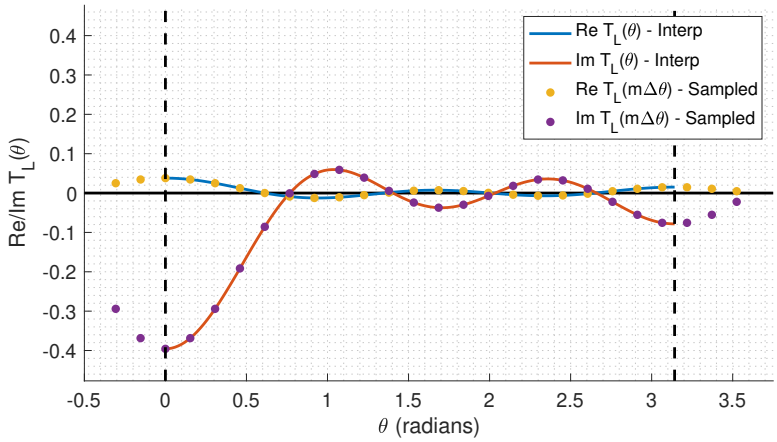


Figure 8.7: Translation operator interpolator. $L = 4$, $s = 5$, $p = 3$, $M = 20$ and there are $M + 2p$ total sampling points. $k = 2\pi$, $r = 10$. Note there is no sampling point at $\theta = \pi$.

The routine `interpTL` takes as inputs the outputs from the preparatory function `interpTLprep` as well as the interpolation point(s). The helper functions are the windowing and Dirichlet kernel, `SN` and `DM`.

```
function [tlinterp] = interpTL(theta,L,p,tlsamp,dth,thetao,M)
% Interpolation function for FMM translation operator
%
% theta:    interpolation points, angle in radians [0 pi]
% L:        truncation degree, sum_{l=0}^L
% s:        over sampling ratio, integer
% p:        number of samples on each side of interpolation point
% Inputs from TLinterppprep:
%   tlsamp:       Translation operator sample points
%   dth:          angular sample spacing, radians
%   thetao:       stepping parameter
%   M:            number of sample points between (0,pi), not including ends
%
% tlinterp:     interpolated translation operator values, size(theta)
%
% Dependencies: interpTLprep, SN, DM

theta = theta(:);
Nt = length(theta);
tlinterp = zeros(Nt,1);
N = M-L;
for n = 1:Nt,
    mo = floor(theta(n)/dth);
    for m=(mo-p+1):(mo+p),
        arg = theta(n)-m*dth;
        tlinterp(n) = tlinterp(n) + tlsamp(m+p)*SN(N,arg,thetao)*DM(M,arg);
    end
end
tlinterp = reshape(tlinterp,size(theta));
```

```
function [tlsamp, dth, thetao, M, theta_samp] = interpTLprep(L,k,r,s,p)
% Preparatory function for translation operator interpolation function TLinterp.
%
% L:    truncation degree, sum_{l=0}^L
% k:    complex wavenumber
% r:    magntude of translation distance
% s:    over sampling ratio, integer
% p:    number of samples on each side fo interpolation point
%
% tlsamp:       Translation operator sample points
% dth:          angular sample spacing, radians
% thetao:       stepping parameter
% M:            number of sample points between (0,pi), not including ends
% theta_samp:   array of theta values
%
% Dependencies: TLth

if L < 0 || s < 0 || p < 0
    disp('bad input')
    return
end
M = s*L;
dth = 2*pi/(2*M+1);
thetao = p*dth;
theta_samp = linspace(-(p-1)*dth,pi-dth/2 + dth*p,M+2*p);
tlsamp = TLth(L,k*r,cos(theta_samp));

function [sn] = SN(N,theta,thetao)
% Windowing function for FMM translation operator interpolation
%
% S_N(theta,theta_o) = R_N(theta,theta_o)/R_N(0,theta_o)
%
% R_N(theta,theta_o) = sinh((2N+1)sinh^-1 sqrt(sin^2(theta_o/2)-sin^2(theta/2)))/...
%                           sqrt(sin^2(theta_o/2)-sin^2(theta/2))
%
% N:        window width parameter
% theta:    interpolation point, angle in radians
% thetao:         stepping parameter
%
% sn:       window function

sqtheta = sqrt(sin(thetao/2).^2 - sin(theta/2).^2);
RNtheta = sinh((2*N+1)*asinh(sqtheta))./sqtheta;
sqo = sin(thetao/2);
RNo = sinh((2*N+1).*asinh(sqo))./sqo;
sn = RNtheta./RNo;
ind = find(theta == -thetao);
if ~isempty(ind)
    sn(ind) = 1;
end

function [dm] = DM(M,theta)
% Dirichlet kernel for FMM translation operator interpolation
%
% D_M(theta) = sin((2M+1)theta/2)((2M+1)sin(theta/2))
%
% M:        number of sample points between (0,pi), not including ends
% theta:    interpolation point, angle in radians
%
% dm:       Dirichlet kernel

tmp = 2*M + 1;
thover2 = theta/2;
dm = sin(tmp*thover2)./(tmp*sin(thover2));
ind = find(theta == 0);
if ~isempty(ind)
    dm(ind) = 1;
end
```

8.6 Scalar Spherical Filter

In this section, we give routines for interpolating and filtering scalar spherical harmonics following [36]. These routines can stand on their own, because they are excellent for general applications of spherical harmonic expansions. The scalar filters can be used in the scalar form of the FMM for interpolating and filtering fields up and down the multi-level hierarchy structure. These lay the ground work for the vector spherical filters derived later.

8.6.1 Spherical Harmonic Transforms

The spherical harmonics form a complete basis, so any band-limited spherical signal can be represented as a finite sum of harmonics

$$f(\theta,\phi)=\sum_{l=0}^{L}\sum_{m=-l}^{l} f_{lm}Y_{lm}(\theta,\phi) \tag{8.51}$$

Spherical harmonics can be written in terms of the normalized Legendre polynomials as

$$Y_{lm}(\theta,\phi)=\frac{1}{\sqrt{2\pi}}\widetilde{P}_l^m(\cos\theta)e^{im\phi} \tag{8.52}$$

Using orthogonality of the spherical harmonics, the expansion coefficients are

$$f_{lm}=\int_0^{2\pi}\int_0^{\pi} f(\theta,\phi)Y^*_{l'm'}(\theta,\phi)\sin\theta d\theta d\phi \tag{8.53}$$

Equation (8.53) is the forward transform, or spherical harmonic analysis, while equation (8.51) is the inverse transform, or spherical harmonic synthesis.

8.6.2 Forward Scalar Spherical Transform

Computing (8.53) consists of two steps: 1) forward Fourier transform in ϕ, 2) forward Legendre transform in θ. Writing out (8.53)

$$f_{lm}=\int_0^{\pi}\widetilde{P}_l^m(\cos\theta)\sin\theta d\theta\frac{1}{\sqrt{2\pi}}\int_0^{2\pi} f(\theta,\phi)e^{-im\phi}d\phi \tag{8.54}$$

where $\widetilde{P}_l^m(\cos\theta)$ are the fully normalized Legendre polynomials. The ϕ integral is computed first in order to create a set of 1D functions of θ for each m

$$f_m(\theta)=\frac{1}{\sqrt{2\pi}}\int_0^{2\pi} f(\theta,\phi)e^{-im\phi}d\phi \tag{8.55}$$

Evaluating this with trapezoidal integration

$$f_m(\theta)=\frac{\sqrt{2\pi}}{I}\sum_{i=1}^{I} f(\theta,\phi_i)e^{-im\phi_i} \tag{8.56}$$

where I is the number of grid points in longitude and $\phi_i=2\pi i/I$ for $i=0,...,I-1$. This can be computed via FFT. The θ integral is next computed for each $f_m(\theta)$ by using the forward Legendre transform with a change of variables

$$\begin{aligned} f_{lm} &= \int_0^{\pi} f_m(\theta)\widetilde{P}_l^m(\cos\theta)\sin\theta d\theta && (8.57)\\ &= -\int_0^{\pi} f_m(\theta)\widetilde{P}_l^m(\cos\theta)d\cos\theta && (8.58)\\ &= \int_{-1}^{1} f_m(\theta(\mu))\widetilde{P}_l^m(\mu)d\mu, \quad \mu=\cos\theta && (8.59)\end{aligned}$$

This can now be evaluated with Gaussian quadrature on the interval $\mu = [-1, 1]$ as

$$f_{lm} = \sum_{j=1}^{J} f_m(\theta_j) \widetilde{P}_l^m(\mu_j) w_j \tag{8.60}$$

where J is the number of points in latitude and the weights, w_j, correspond to the nodes $\theta_j = \arccos \mu_j$. One first selects the number of grid points in latitude, retrieves the Gaussian nodes for that number of integration points, then evaluates the points θ_j. Because this operation is integrating products of spherical harmonics, the integral is exact if the number of grid points in latitude and longitude are $J = L + 1$ and $I = 2L + 1$ for coefficients through L. By virtue of the Gaussian quadrature node spacing, the field is never evaluated at the poles.

The routine `sst` performs the forward scalar spherical transform and returns the spectral coefficients f_{lm}. The coefficients are returned on a 1D array of size $L^2 + 2L + 1$, linearly indexed. It takes as inputs the maximum degree L for which harmonics are desired. The spherical function $f(\theta_j, \phi_i)$ is sampled on an $I \times J$ meshgrid, where $I = 2L' + 1$ and $J = L' + 1$ are such that $L' \geq L$. The sample points need to be $\phi_i = 2\pi i/I$ for $i = 0, ..., I - 1$, and $\theta_j = \arccos \mu_j$, where μ_j are the J quadrature nodes on $\mu = [-1, 1]$, which are also inputs. In other words, the grid can be sampled more finely than the maximum degree of the harmonics desired for the coefficients. $f_m(\theta_j)$ is computed in place with an FFT along the first dimension of the array. Matlab's `fft` produces a two-sided DFT, and $2L + 1$ is always odd, so the rows of the matrix $f_m(\theta_j)$ correspond to the spectral components $m = 0, 1, ..., (I-1)/2, -(I-1)/2, ..., -1$. The rows of the 1D FFT are indexed

$$\text{idx}(I, m) = \begin{cases} m + 1, & m \geq 0 \\ I - m + 1, & m < 0 \end{cases} \tag{8.61}$$

```
function [flm] = sst(f,L,muj,wj)
% Forward scalar spherical transform
%
% f:    [IxJ] sampled spherical function
%       I = 2*L'+1, J = L'+1
%       phi_i = 2*pi*(0:(I-1))/I
%       theta = arccos(mu_j)
% L:    maximum degree L for output coefficients flm (L' >= L)
% muj:  J Gaussian quadrature nodes
% wj:   J Gaussian quadrature weights
%
% flm:  [Nx1], N = L^2+2*L+1, spectral coefficients, linearly indexed
%
% Dependencies: Plm

[I, J] = size(f);
tot = L^2 + 2*L + 1;
flm = zeros(tot,1);
% phi transform
fmth = (sqrt(2*pi)/I)*fft(f,[],1);
% Legendre polynomials evaluatd at muj
plm = Plm(L,muj);
% theta quadrature integral
wj = wj(:).';
for l=0:L,
for m=-l:l,
    ind = lm2ind(l,m,'mono');
    if m >= 0
        ind2 = m + 1;
    else
        ind2 = I + m + 1;
    end
    flm(ind) = sum(fmth(ind2,:).*plm(ind,:).*wj);
end
end
```

8.6.3 Inverse Scalar Spherical Transform

The inverse transform consists of taking the expansion coefficients f_{lm} and applying 1) the inverse Legendre transform in θ, 2) the inverse Fourier transform in ϕ. The inverse Legendre transform is

$$f_m(\theta_j) = \sum_{l=|m|}^{L} f_{lm} \widetilde{P}_l^m(\mu_j) \tag{8.62}$$

where again $\mu_j = \cos\theta_j$. The inverse Fourier transform in ϕ is

$$f(\theta_j, \phi_i) = \frac{1}{\sqrt{2\pi}} \sum_{m=-L}^{L} f_m(\theta_j) e^{im\phi_i} \tag{8.63}$$

The routine `isst` computes the inverse scalar spherical transform. The inputs are the array of harmonics f_{lm} of size $L^2 + 2L + 1$ linearly indexed, and the maximum degree L. It then returns the $I \times J$ spherical function $f(\theta_j, \phi_i)$ as a meshgrid such that $I = 2L' + 1$ and $J = L' + 1$, where $\phi_i = 2\pi i/I$ for $i = 0, ..., I-1$, and $\mu_j = \cos\theta_j$. J is determined by the length of the input μ_j, which can be larger than the corresponding sampling of the L harmonics in f_{lm} (this allows the routine to preform interpolation automatically). An additional factor of I is needed because Matlab's `ifft` divides by the number of samples in ϕ.

```
function [f] = isst(flm,L,muj)
% Inverse scalar spherical transform
%
% flm:  tot = L^2+2*L+1 spectral coefficients
% L:    maximum degree L
% muj:  J Gaussian quadrature nodes, muj = cos(theta_j)
%
% f:    [IxJ] gridded spherical function
%       I = 2*L'+1, J = L'+1, where J = length(muj)
%
% Dependencies: Plm

J = length(muj);
I = 2*(J-1) + 1;
% Legendre polynomials evaluatd at muj
plm = Plm(L,muj);
% inverse Legendre transform
fmth = zeros(I,J);
for m = -L:L,
    if m >= 0
        ind2 = m + 1;
    else
        ind2 = I + m + 1;
    end
    for l = abs(m):L,
        ind = lm2ind(l,m,'mono');
        fmth(ind2,:) = fmth(ind2,:) + flm(ind)*plm(ind,:);
    end
end
% inverse Fouriner transform in phi
f = (I/sqrt(2*pi))*ifft(fmth,[],1);
```

8.6.4 Scalar Spherical Filter

The forward and inverse scalar spherical transforms can be used together to accomplish interpolation or filtering (anterpolation) of a spherical function.

Interpolation takes a function $f(\theta,\phi)$ with coarse sampling and L harmonics, and upsamples it to a function $f(\theta',\phi')$ with finer sampling and K harmonics, where $K > L$. Because the original signal is band limited with maximum harmonic L, the interpolated signal contains the same harmonic content, and is interpolated exactly. This is the spherical harmonic analog of upsampling a Nyquist-sampled exactly to an arbitrarily fine sampling with sinc interpolation. Interpolation is accomplished by first computing the spectral components f_{lm} of $f(\theta,\phi)$ using the scalar spherical transform to degree L, zero-padding the coefficients to degree K to form f'_{lm}, then applying the inverse scalar spherical transform to create $f(\theta',\phi')$.

Filtering takes $f(\theta',\phi')$ with fine sampling and K harmonics to a function $f(\theta,\phi)$ with coarse sampling and L harmonics, where $L < K$. This is analogous to filtering a signal and resampling it at lower rate. Filtering necessarily eliminates higher frequency harmonics. Filtering is accomplished by computing the spectral components f'_{lm} of $f(\theta',\phi')$ via the SST to degree K, truncating down to degree L to form f_{lm}, then applying the ISST to create $f(\theta,\phi)$.

$$\begin{array}{rccccccccc}
\text{Interpolation:} & f(\theta,\phi) & \overset{\text{sst}}{\rightarrow} & f_{lm}, L & \rightarrow & \text{zero pad} & \rightarrow & f'_{lm}, K & \overset{\text{isst}}{\rightarrow} & f(\theta',\phi') \\
\text{Filter:} & f(\theta',\phi') & \overset{\text{sst}}{\rightarrow} & f'_{lm}, K & \rightarrow & \text{trunctate} & \rightarrow & f_{lm}, L & \overset{\text{isst}}{\rightarrow} & f(\theta,\phi)
\end{array}$$

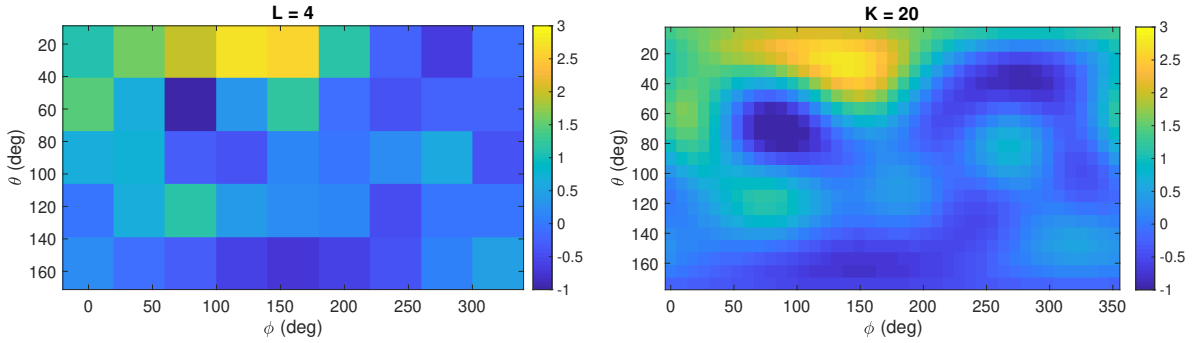


Figure 8.8: Interpolation of a complex scalar field (real part). Left: coarsely sampled field. Right: finely sampled interpolated field.

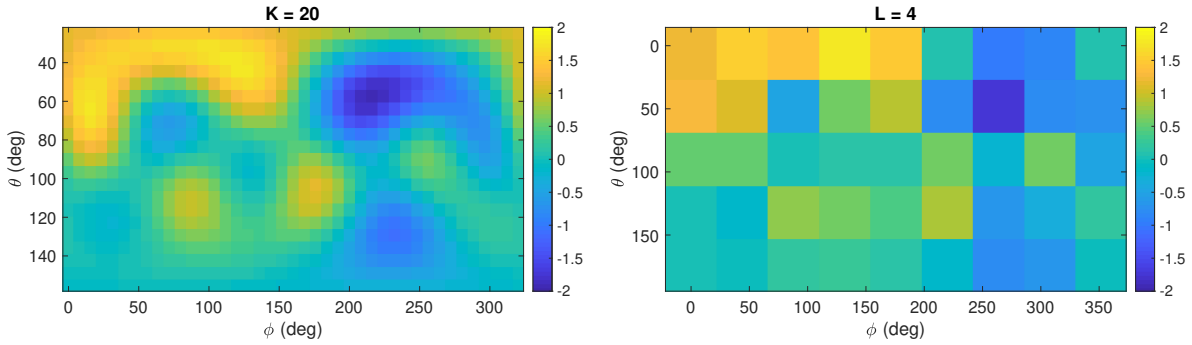


Figure 8.9: Filtering of a complex scalar field (real part). Left: finely sampled field. Right: coarsely sampled filtered field.

The routine `ssfilt` computes the scalar spherical interpolation or filtering operation. It takes the maximum harmonic degrees L and K, where $L \leq K$. The harmonic content is either interpolated from

L to K or filtered from K to L. For interpolation, the input function is $f(\theta, \phi)$, sized $2L'+1 \times L'+1$ on a meshgrid, and the routine returns $f(\theta', \phi')$, sized $2K'+1 \times K'+1$, where L' and K' are set by the length of μ_j and μ_k, respectively. The sampling of the grid can have more harmonics than the interpolation/filter harmonics. As always, it assumes ϕ is uniformly spaced and θ is spaced according to the Gaussian quadrature nodes. This routine calls `sst` and `isst` that recompute the underlying Legendre polynomials at each run and is not built for speed.

```
function [f2] = ssfilt(f,L,muj,wj,K,muk,wk)
% Scalar spherical filter (interpolation or filtering)
% Assumes L <= K
%
% L:    maximum degree L<=L'
% muj:  Quadrature nodes for L'+1 points
% wj:   Quadrature weights for L'+1 ponts
% K:    maximum degree K<=K'
% muk:  Quadrature nodes for K'+1 points
% wk:   Quadrature weights for K'+1 points

% Interpolation
% f:    input scalar field f(theta_j,phi_j):    (2L'+1)x(L'+1)
% f2:   output scalar field f(theta_k,phi_k): (2K'+1)x(K'+1)
%
% Filter
% f:    input scalar field f(theta_k,phi_k):  (2K'+1)x(K'+1)
% f2:   output scalar field f(theta_j,phi_j):   (2L'+1)x(L'+1)
%
% Dependencies: sst, isst

if L > K
    error('error, L > K')
end
if L == K
    f2 = f;
    return
end
[M N] = size(f);
J = length(muj);
I = 2*(J-1) + 1;
Q = length(muk);
P = 2*(Q-1) + 1;
if M == I % interpolation mode
    if N ~= J
        error('bad array size')
    end
    filt = 0;
elseif M == P; % filter mode
    if N ~= Q
        error('bad array size')
    end
    filt = 1;
else
    error('bad array size')
end
tot = L^2 + 2*L + 1;
if ~filt % interpolation
    flm = sst(f,L,muj,wj);
    tot2 = K^2 + 2*K + 1;
    flm2 = zeros(tot2,1);
    flm2(1:tot) = flm;
    f2 = isst(flm2,K,muk);
else % filter
    flm = sst(f,K,muk,wk);
    flm2 = flm(1:tot);
    f2 = isst(flm2,L,muj);
end
```

8.6.5 Fast Scalar Spherical Filter

The bottle neck of the scalar spherical filter lies in the sums of the forward and inverse Legendre transforms, especially when L is large. The fix is to combine the two transforms, after which the sums can be simplified and further accelerated with the 1D FMM, which is described in Section 8.9. For a discussion of the computational complexity, see [36].

Assume that interpolation is being done and $L < K$. The quantities θ_j, μ_j, and w_j are associated with L harmonics of field $f(\theta_j, \phi_i)$, and the quantities θ_k, μ_k, and w_k are associated with K harmonics of field $f'(\theta_k, \phi_k)$ and the routine is interpolating $f(\theta_j, \phi_i)$ to $f'(\theta_k, \phi_k)$. Start by substituting the forward Legendre transform, (8.60), into the inverse Legendre transform, (8.62),

$$f'_m(\theta_k) = \sum_{l=|m|}^{K} \left(\sum_{j=1}^{J} f_m(\theta_j) \widetilde{P}_l^m(\mu_j) w_j \right) \widetilde{P}_l^m(\mu_k) \tag{8.64}$$

When $L < K$, the sum over K can be restricted to L because $f_m(\theta_j)$ does not have harmonics when $|m| > L$. This is equivalent to zero-padding the harmonics f_{lm} in the standard spherical filter. Changing the limit of the sum and exchanging the order of the sums

$$f'_m(\theta_k) = \sum_{j=1}^{J} f_m(\theta_j) w_j \sum_{l=|m|}^{L} \widetilde{P}_l^m(\mu_j) \widetilde{P}_l^m(\mu_k) \tag{8.65}$$

The sum over L can be simplified with the Christoffel-Darboux formula

$$\sum_{l=|m|}^{L} \widetilde{P}_l^m(\mu_j) \widetilde{P}_l^m(\mu_k) = \epsilon_{L+1}^m \frac{\widetilde{P}_{L+1}^m(\mu_k) \widetilde{P}_L^m(\mu_j) - \widetilde{P}_L^m(\mu_k) \widetilde{P}_{L+1}^m(\mu_j)}{\mu_k - \mu_j} \tag{8.66}$$

where

$$\epsilon_l^m = \sqrt{\frac{l^2 - m^2}{4l^2 - 1}} \tag{8.67}$$

Substituting (8.66) into (8.65) and separating terms

$$\frac{f'_m(\theta_k)}{\epsilon_{L+1}^m} = \widetilde{P}_{L+1}^m(\mu_k) \sum_{j=1}^{J} \frac{f_m(\theta_j) w_j \widetilde{P}_L^m(\mu_j)}{\mu_k - \mu_j} - \widetilde{P}_L^m(\mu_k) \sum_{j=1}^{J} \frac{f_m(\theta_j) w_j \widetilde{P}_{L+1}^m(\mu_j)}{\mu_k - \mu_j} \tag{8.68}$$

This has the form of a matrix-vector multiply over the kernel of type $1/(x - x')$, therefore it is possible to use the 1D FMM to accelerate this computation.

When the sum rolls over the case $\mu_k = \mu_j$, L'Hopital's rule can be applied to either μ_k or μ_j to resolve the singularity. The condition $\mu_k = \mu_j$ only ever occurs at $\mu = 0$, because the nodes of quadrature for different degrees of harmonics never overlap expect at $\mu = 0$, and only when the number of quadrature points in θ of both functions is odd. Assuming the number of quadrature points is equal to $L + 1$ and $K + 1$, this means that the rule needs to be applied when both L and K are even and different (when $L = K$ there is nothing to interpolate or filter). It is possible to avoid the singularity entirely by requiring that L and K always be odd, then $\mu_j \neq \mu_k$ and the Legendre derivative are not needed, but this is too restrictive. Applying L'Hopital's rule to μ_j we get a version of (8.68) that handles the singularity

$$\begin{aligned} \frac{f'_m(\theta_k)}{\epsilon_{L+1}^m} \quad = \quad & \widetilde{P}_{L+1}^m(\mu_k) \sum_{j=1}^{J} f_m(\theta_j) w_j \begin{cases} \dfrac{\widetilde{P}_L^m(\mu_j)}{\mu_k - \mu_j} & \mu_j \neq \mu_k \\ -\dfrac{d\widetilde{P}_L^m(\mu_j)}{d\mu_j} & \mu_j = \mu_k \end{cases} \\ & - \widetilde{P}_L^m(\mu_k) \sum_{j=1}^{J} f_m(\theta_j) w_j \begin{cases} \dfrac{\widetilde{P}_{L+1}^m(\mu_j)}{\mu_k - \mu_j} & \mu_j \neq \mu_k \\ -\dfrac{d\widetilde{P}_{L+1}^m(\mu_j)}{d\mu_j} & \mu_j = \mu_k \end{cases} \end{aligned} \tag{8.69}$$

Another way to think about the singular point, when it occurs, is to consider $1/(\mu_k - \mu_j)$ as a matrix that is multiplied on the right by a vector that indexes μ_j (e.g., $f_m(\theta_j) w_j \widetilde{P}_L^m(\mu_j)$) and multiplied element-wise on the left by a vector that indexes μ_k (e.g., $\widetilde{P}_L^m(\mu_k)$). Only the central matrix element needs to be adjusted, but the adjustment applies to both the matrix element and the element in the right hand vector. This makes what should be a simple matrix-vector multiply awkward to compute. We handle this by recomputing the row-vector multiplication that contains the singular point separately, but better solutions exist. Finally, the application of L'Hopital's rule in [36] does not appear to handle the sum over j correctly, and [44] mentions this procedure but does not give the equations.

The above equations are for interpolation. For filtering, simply exchange the nodes and weights between L and K. The intermediate sum will be restricted again to a maximum degree L, because the filtered field only has harmonics up to order $m = L$. Truncating the intermediate sum is equivalent to truncating the coefficients f'_{lm} in the standard spherical filter.

Basic implementation

The routine `fssfilt` implements a basic version of the fast scalar spherical filter and works the same as `ssfilt`. It expects $L \leq K$ and that the input field be sampled at the nodes of quadrature with either $I = 2L' + 1$ and $J = L' + 1$ points for interpolation, or $P = 2K' + 1$ and $Q = K' + 1$ points for filtering. L' and K' are set by the length of μ_j and μ_k, respectively, such that $L' \geq L$ or $K' \geq K$. Provisions are included for the singular point based on the values of L' and K'. It computes the matrix-vector multiplication directly and does not implement the 1D FMM acceleration. It also computes the Legendre polynomials anew at each call, so it can be made much faster with appropriate precomputation, because only the L and $L+1$ harmonics are needed. The routine returns the same result as `ssfilt` to machine precision, and is several times faster for low number of harmonics.

```
function [f2] = fssfilt(f,L,muj,wj,K,muk,wk)
% Fast scalar spherical filter, basic implementation
% Assumes L <= K
%
% L:    maximum degree L<=L'
% muj:  Quadrature nodes for L'+1 points
% wj:   Quadrature weights for L'+1 ponts
% K:    maximum degree K<=K'
% muk:  Quadrature nodes for K'+1 points
% wk:   Quadrature weights for K'+1 points

% Interpolation
% f:    input scalar field f(theta_j,phi_j):    (2L'+1)x(L'+1)
% f2:   output scalar field f(theta_k,phi_k): (2K'+1)x(K'+1)
%
% Filter
% f:    input scalar field f(theta_k,phi_k):  (2K'+1)x(K'+1)
% f2:   output scalar field f(theta_j,phi_j):   (2L'+1)x(L'+1)
%
% Dependencies: Plm, Plmp

if L > K
    disp('error, L > K')
    return
end
if L == K
    f2 = f;
    return
end
[M, N] = size(f);
J = length(muj);
I = 2*(J-1) + 1;
Q = length(muk);
P = 2*(Q-1) + 1;
if M == I % interpolation mode
    if N ~= J
        disp('bad array size')
        return
```

```
    end
    filt = 0;
    const = P/I;
    fmthp = zeros(P,Q);
    ind_sing = (Q+1)/2;
    sing_lim = J;
elseif M == P % filter mode
    if N ~= Q
        disp('bad array size')
        return
    end
    filt = 1;
    const = I/P;
    fmthp = zeros(I,J);
    % swap the quadrature nodes if filtering
    tmpmuj = muj;
    tmpwj = wj;
    muj = muk;
    wj = wk;
    muk = tmpmuj;
    wk = tmpwj;
    ind_sing = (J+1)/2;
    sing_lim = Q;
else
    disp('bad array size')
    return
end

% compute Legendre polynomials
PLj = Plm(L+1,muj);
PLk = Plm(L+1,muk);
PLjp = Plmp(L+1,muj);

% compute 1/(\mu_k - \mu_j) matrix
[Uk, Uj] = meshgrid(muk,muj);
M = 1./(Uk - Uj);

% if P and Q are odd, take care of singularity later
singularity = mod(J,2) && mod(Q,2);

% phi transform
fmth = fft(f,[],1);

% Combined forward/inverse Legendre transforms
indLmbase = L^2 + L + 1;
indLp1mbase = (L+1)^2 + (L+1) + 1;
for m=-L:L,
    indLm = indLmbase + m;
    indLp1m = indLp1mbase + m;
    if m >= 0
        indk = m + 1;
        indj = m + 1;
    else
        if ~filt
            indk = P + m + 1;
            indj = I + m + 1;
        else
            indk = I + m + 1;
            indj = P + m + 1;
        end
    end

    % terms of the sum
    bj = fmth(indj,:).*(wj).*PLj(indLm,:);
    bjp1 = fmth(indj,:).*(wj).*PLj(indLp1m,:);

    % matrix-vector multiply
    coefj = bj*M;
```

```
    coefjp1 = bjp1*M;

    % recompute one sum when the singularity exists
    if singularity
        tmp1 = 0;
        tmp2 = 0;
        for n=1:sing_lim,
            c1 = fmth(indj,n)*wj(n);
            if n == (sing_lim+1)/2;
                tmp1 = tmp1 - c1*PLjp(indLm,n);
                tmp2 = tmp2 - c1*PLjp(indLp1m,n);
            else
                tmp1 = tmp1 + c1*PLj(indLm,n)/(muk(ind_sing)-muj(n));
                tmp2 = tmp2 + c1*PLj(indLp1m,n)/(muk(ind_sing)-muj(n));
            end
        end
        coefj(ind_sing) = tmp1;
        coefjp1(ind_sing) = tmp2;
    end
    bk = PLk(indLm,:);
    bkp1 = PLk(indLp1m,:);
    epsilon = sqrt(((L+1)^2 - m^2)/(4*(L+1)^2 - 1));
    fmthp(indk,:) = epsilon*(bkp1.*coefj - bk.*coefjp1);
end

% inverse phi transform
f2 = const*ifft(fmthp,[],1);
```

8.7 Vector Spherical Filter

In this section, we give routines for interpolating and filtering vector spherical harmonics. Like the scalar routines, they could also stand on their own apart from the FMM. Fast versions of the vector spherical filter are also possible, either based on similar concepts of the fast scalar filter in which the forward and inverse Legendre transforms are compressed, or by using the fast scalar filter with modifications.

8.7.1 Vector Spherical Harmonic Transforms

The vector spherical harmonics form a complete basis, so any band limited vector field can be represented as sum of harmonics

$$\begin{aligned}\mathbf{F}(\theta,\phi) &= F_\theta(\theta,\phi)\hat{\theta} + F_\phi(\theta,\phi)\hat{\phi} &(8.70)\\ &= \sum_{l=1}^{L}\sum_{m=-l}^{l} b_{lm}\mathbf{B}_{lm}(\theta,\phi) + c_{lm}\mathbf{C}_{lm}(\theta,\phi) &(8.71)\end{aligned}$$

where $F_\theta(\theta,\phi)$ and $F_\phi(\theta,\phi)$ are scalar spherical functions representing each vector component. In general, the vector spherical harmonics, $\mathbf{B}_{lm}$ and $\mathbf{C}_{lm}$, could be fully normalized or partially normalized. Eventually, the fast vector spherical filter will use the fast scalar filter and, to accommodate this, it is best to use the partially normalized vector spherical harmonics in the derivations that follow (as opposed to the fully normalized versions that include a factor of $1/\sqrt{l(l+1)}$). The scalar functions are expanded as

$$\begin{aligned}F_\theta(\theta,\phi) &= \sum_{l=1}^{L}\sum_{m=-l}^{l} b_{lm}\frac{d}{d\theta}Y_{lm}(\theta,\phi) + c_{lm}\frac{im}{\sin\theta}Y_{lm}(\theta,\phi) &(8.72)\\ F_\phi(\theta,\phi) &= \sum_{l=1}^{L}\sum_{m=-l}^{l} b_{lm}\frac{im}{\sin\theta}Y_{lm}(\theta,\phi) - c_{lm}\frac{d}{d\theta}Y_{lm}(\theta,\phi) &(8.73)\end{aligned}$$

which show the mixing of harmonics between vector components.
The orthogonality relations for these partially normalized vector spherical harmonics are

$$\begin{aligned}\int_0^{2\pi}\int_0^{\pi}\left\{\begin{array}{l}\mathbf{B}_{lm}(\theta,\phi)\cdot\mathbf{B}^*_{lm}(\theta,\phi)\\ \mathbf{C}_{lm}(\theta,\phi)\cdot\mathbf{C}^*_{lm}(\theta,\phi)\end{array}\right\}\sin\theta d\theta d\phi &= l(l+1)\delta_{ll'}\delta_{mm'} &(8.74)\\ \int_0^{2\pi}\int_0^{\pi}\left\{\begin{array}{l}\mathbf{B}_{lm}(\theta,\phi)\cdot\mathbf{C}^*_{lm}(\theta,\phi)\end{array}\right\}\sin\theta d\theta d\phi &= 0 &(8.75)\end{aligned}$$

Given a vector field $\mathbf{F}(\theta,\phi)$, the coefficients are found with

$$\left\{\begin{array}{l}b_{lm}\\ c_{lm}\end{array}\right\} = \frac{1}{l(l+1)}\int_0^{2\pi}\int_0^{\pi}\mathbf{F}(\theta,\phi)\cdot\left\{\begin{array}{l}\mathbf{B}^*_{lm}(\theta,\phi)\\ \mathbf{C}^*_{lm}(\theta,\phi)\end{array}\right\}\sin\theta d\theta d\phi \tag{8.76}$$

8.7.2 Forward Vector Spherical Transform

The forward vector spherical transform, as in the scalar case, is composed of a forward Fourier transform and forward Legendre transform. Writing out (8.76)

$$\begin{aligned}b_{lm} &= \frac{1}{l(l+1)}\int_0^{2\pi}\int_0^{\pi}\frac{1}{\sqrt{2\pi}}\left(F_\theta(\theta,\phi)\frac{\partial\widetilde{P}_l^m(\cos\theta)}{\partial\theta}e^{-im\phi}\right)\sin\theta d\theta d\phi\\ &\quad+\frac{1}{l(l+1)}\int_0^{2\pi}\int_0^{\pi}\frac{1}{\sqrt{2\pi}}\left(F_\phi(\theta,\phi)\frac{(-im)}{\sin\theta}\widetilde{P}_l^m(\cos\theta)e^{-im\phi}\right)\sin\theta d\theta d\phi &(8.77)\end{aligned}$$

$$c_{lm} = \frac{1}{l(l+1)}\int_0^{2\pi}\int_0^{\pi}\frac{1}{\sqrt{2\pi}}\left(F_\theta(\theta,\phi)\frac{(-im)}{\sin\theta}\widetilde{P}_l^m(\cos\theta)e^{-im\phi}\right)\sin\theta d\theta d\phi$$
$$-\frac{1}{l(l+1)}\int_0^{2\pi}\int_0^{\pi}\frac{1}{\sqrt{2\pi}}\left(F_\phi(\theta,\phi)\frac{\partial\widetilde{P}_l^m(\cos\theta)}{\partial\theta}e^{-im\phi}\right)\sin\theta d\theta d\phi \quad (8.78)$$

The integrals over latitude and longitude can be separated. Performing the ϕ integral first we have

$$\begin{Bmatrix} f_{\theta,m}(\theta) \\ f_{\phi,m}(\theta) \end{Bmatrix} = \frac{1}{\sqrt{2\pi}}\int_0^{\pi}\begin{Bmatrix} F_\theta(\theta,\phi) \\ F_\phi(\theta,\phi) \end{Bmatrix}e^{-im\phi}d\phi \quad (8.79)$$
$$= \frac{\sqrt{2\pi}}{I}\sum_{i=1}^{I}\begin{Bmatrix} F_\theta(\theta,\phi_i) \\ F_\phi(\theta,\phi_i) \end{Bmatrix}e^{-im\phi_i} \quad (8.80)$$

where the grid points are $\phi_i = 2\pi i/I$ for $i = 0, ..., I-1$. These are evaluated with a fast Fourier transform. The coefficients are then written in terms of $f_{\theta,m}(\theta)$ and $f_{\phi,m}(\theta)$ as

$$b_{lm} = \frac{1}{l(l+1)}\int_0^{\pi}\left(\frac{\partial\widetilde{P}_l^m(\cos\theta)}{\partial\theta}f_{\theta,m}(\theta) + \frac{(-im)}{\sin\theta}\widetilde{P}_l^m(\cos\theta)f_{\phi,m}(\theta)\right)\sin\theta d\theta \quad (8.81)$$

$$c_{lm} = \frac{1}{l(l+1)}\int_0^{\pi}\left(\frac{(-im)}{\sin\theta}\widetilde{P}_l^m(\cos\theta)f_{\theta,m}(\theta) - \frac{\partial\widetilde{P}_l^m(\cos\theta)}{\partial\theta}f_{\phi,m}(\theta)\right)\sin\theta d\theta \quad (8.82)$$

The integrations are performed exactly with Gaussian quadrature after a change of variables. The first change of variables is of the type

$$\int_0^{\pi}\frac{1}{\sin\theta}f_m(\theta)\widetilde{P}_l^m(\cos\theta)\sin\theta d\theta = -\int_0^{\pi}\frac{1}{\sin\theta}f_m(\theta)\widetilde{P}_l^m(\cos\theta)d\cos\theta \quad (8.83)$$
$$= \int_{\pi}^{0}\frac{1}{\sqrt{1-\cos^2\theta}}f_m(\theta)\widetilde{P}_l^m(\cos\theta)d\cos\theta \quad (8.84)$$
$$= \int_{-1}^{1}\frac{1}{\sqrt{1-\mu^2}}f_m(\theta(\mu))\widetilde{P}_l^m(\mu)d\mu, \quad \mu = \cos\theta \quad (8.85)$$

The second change of variables is of the type

$$\int_0^{\pi}f_m(\theta)\frac{\partial\widetilde{P}_l^m(\cos\theta)}{\partial\theta}\sin\theta d\theta = -\int_0^{\pi}f_m(\theta)\frac{\partial\widetilde{P}_l^m(\cos\theta)}{\partial\theta}d\cos\theta \quad (8.86)$$
$$= \int_{\pi}^{0}f_m(\theta)\frac{\partial\widetilde{P}_l^m(\cos\theta)}{\partial\theta}d\cos\theta \quad (8.87)$$
$$= -\int_{-1}^{1}\sqrt{1-\mu^2}f_m(\theta(\mu))\frac{\partial\widetilde{P}_l^m(\mu)}{\partial\mu}d\mu, \quad \mu = \cos\theta \quad (8.88)$$

where we have used the chain rule

$$\frac{\partial\widetilde{P}_l^m(\cos\theta)}{\partial\theta} = \frac{\partial\widetilde{P}_l^m(\cos\theta)}{\partial\mu}\frac{\partial\mu}{\partial\theta} = \frac{\partial\widetilde{P}_l^m(\cos\theta)}{\partial\mu}(-\sin\theta) = -\sqrt{1-\mu^2}\frac{\partial\widetilde{P}_l^m(\mu)}{\partial\mu}$$

Note, in [36], the equations are given in terms of $\partial \widetilde{P}_l^m(\mu_j)/\partial\theta$ and the chain rule is not applied. The chain rule is required for the derivatives to be compatible with our computations of the Legendre derivatives.

Equations (8.81) and (8.82) can now be evaluated via Gaussian quadrature on the interval $\mu = [-1, 1]$ as

$$b_{lm} = \frac{1}{l(l+1)} \sum_{j=1}^{J} \left(\left(-\sqrt{1-\mu_j^2}\right) \frac{\partial \widetilde{P}_l^m(\mu_j)}{\partial \mu} f_{\theta,m}(\theta_j) + \frac{(-im)}{\sqrt{1-\mu_j^2}} \widetilde{P}_l^m(\mu_j) f_{\phi,m}(\theta_j) \right) w_j \tag{8.89}$$

$$c_{lm} = \frac{1}{l(l+1)} \sum_{j=1}^{J} \left(\frac{(-im)}{\sqrt{1-\mu_j^2}} \widetilde{P}_l^m(\mu_j) f_{\theta,m}(\theta_j) - \left(-\sqrt{1-\mu_j^2}\right) \frac{\partial \widetilde{P}_l^m(\mu_j)}{\partial \mu} f_{\phi,m}(\theta_j) \right) w_j \tag{8.90}$$

where J is the number of integration points in longitude with weights w_j and Gaussian nodes $\mu_j = \cos\theta_j$.

The routine `vst` takes as input the scalar functions $F_\theta(\theta, \phi)$ and $F_\phi(\theta, \phi)$ sampled such that the number of rows is $I = 2L + 1$ and number of columns is $J = L + 1$ sampled at the points of quadrature. It returns the expansion coefficients b_{lm} and c_{lm} linearly indexed. It is otherwise similar in form to the routine for the scalar spherical transform, `sst`, except that there is no monopole component. The routine defaults to the partially normalized vector spherical harmonics as derived above. For fully normalized harmonics, use optional string switch `norm` that will use a factor of $1/\sqrt{l(l+1)}$, and `none` for no factors of l. The Legendre polynomials can be optionally precomputed for repeated application over fields of the same sampling.

```
function [blm, clm] = vst(Fth,Fphi,L,muj,wj,normstr,plm,dplm)
% Forward vector spectral transform
% Defaults to partially normalized vector spherical hamornics
%
% L:        maximum degree L<=L'
% Fth:      IxJ sampled theta component F_theta(theta,phi)
% Fphi:     IxJ sampled phi component F_phi(theta,phi)
%           I = 2L'+1, J = L'+1
%           phi_i = 2*pi*(0:(I-1))/I
%           theta = arccos(mu_j)
% muj:      J Gaussian quadrature nodes
% wj:       J Gaussian quadrature weights
% normstr:  [optional] 'norm' for fully normalized spherical harmonics
%                      'none' for no normalization
% plm,dplm: [optional] precomputed Legendre polynomials
%           plm = Plm(L,muj);
%           dplm = Plmp(L,muj);
%
% blm,clm:  L^2+2*L spectral coefficients, linearly indexed
%
% dependencies: Plm, Plmp

[I, J] = size(Fth);
tot = L^2 + 2*L;

% fft in \phi
fthm = sqrt(2*pi)/I*fft(Fth,[],1);
fphim = sqrt(2*pi)/I*fft(Fphi,[],1);

% Legendre polynomials and derivative
if nargin <= 6
    plm = Plm(L,muj); % includes monopole
    dplm = Plmp(L,muj); % includes monopole
elseif nargin == 8 && ~exist('plm') && ~exist('dplm')
    disp('bad inputs for precomputation')
    return
end

muj = muj(:)';
```

```
wj = wj(:)';
sq = sqrt(1-muj.^2);
blm = zeros(tot,1);
clm = zeros(tot,1);
for l=1:L,
for m=-l:l,
    ind = l^2 + l + m;
    c1 = -sq.*dplm(ind+1,:); % monopole indexing
    c2 = -1i*m*plm(ind+1,:)./sq; % monopole indexing
    if m >= 0
        ind2 = m + 1;
    else
        ind2 = I + m + 1;
    end
    if nargin >= 6 && strcmp(normstr,'none')
        const = 1;
    elseif nargin >= 6 && strcmp(normstr,'norm')
        const = 1/sqrt(l*(l+1));
    else
        const = 1/(l*(l+1));
    end
    blm(ind) = const*sum((c1.*fthm(ind2,:) + c2.*fphim(ind2,:)).*wj);
    clm(ind) = const*sum((c2.*fthm(ind2,:) - c1.*fphim(ind2,:)).*wj);
end
end
```

8.7.3 Inverse Vector Spherical Transform

Given coefficients b_{lm} and c_{lm} the inverse vector spherical transform is computed by first applying the inverse Legendre transform then an inverse Fourier transform. Note, the factor of $l(l+1)$ is not needed for partially normalized vector spherical wave functions.

$$f_{\theta,m}(\theta_j) = \sum_{l=|m|}^{L} b_{lm}\left(-\sqrt{1-\mu_j^2}\right)\frac{\partial \widetilde{P}_l^m(\mu_j)}{\partial \mu} + c_{lm}\frac{im}{\sqrt{1-\mu_j^2}}\widetilde{P}_l^m(\mu_j) \tag{8.91}$$

$$f_{\phi,m}(\theta_j) = \sum_{l=|m|}^{L} b_{lm}\frac{im}{\sqrt{1-\mu_j^2}}\widetilde{P}_l^m(\mu_j) - c_{lm}\left(-\sqrt{1-\mu_j^2}\right)\frac{\partial \widetilde{P}_l^m(\mu_j)}{\partial \mu} \tag{8.92}$$

Again, the Gaussian nodes are $\mu_j = \cos\theta_j$. The inverse Fourier transform of $f_{\theta,m}(\theta_j)$ and $f_{\phi,m}(\theta_j)$ in ϕ then gives

$$\left\{\begin{array}{c} F_\theta(\theta_j,\phi_i) \\ F_\phi(\theta_j,\phi_i) \end{array}\right\} = \frac{1}{\sqrt{2\pi}}\sum_{m=-L}^{L}\left\{\begin{array}{c} f_{\theta,m}(\theta_j) \\ f_{\phi,m}(\theta_j) \end{array}\right\} e^{im\phi_i} \tag{8.93}$$

The routine `ivst` computes the inverse vector spherical transform given coefficients b_{lm} and c_{lm}. It returns the vector field components $F_\theta(\theta,\phi)$ and $F_\phi(\theta,\phi)$. The coefficients matrices contain all harmonics up through L all m and must be length L^2+2L. There has to be at least $I = 2L+1$ sampling points in ϕ and at least $J = L+1$ nodes of quadrature in θ, which is determined from the length of the input μ_j. Like `isst`, this allows the routine to performs interpolation automatically onto a grid that is sampled for a harmonic degree larger than L. The routine defaults to the partially normalized vector spherical harmonics. For fully normalized harmonics, use optional string switch `norm` to include a factor of $1/\sqrt{l(l+1)}$. The Legendre polynomials can be optionally precomputed for repeated application over fields of the same sampling.

```
function [Fth,Fphi] = ivst(blm,clm,L,muj,normstr,plm,dplm)
% Inverse vector spectral transform
%
% L:        Maximum degree L<=L'
```

```
% blm,clm:  L^2 + 2*L spectral coefficients on B_{lm} and C_{lm}
% muj:      J Gaussian quadrature nodes, muj = cos(theta_j) (L' = J-1 harmonics)
% normstr:  [optional] 'norm' for fully normalized spherical harmonics
% plm,dplm: [optional] precomputed Legendre polynomials
%           plm = Plm(L,muj);
%           dplm = Plmp(L,muj);
%
% Fth:      I'xJ' sampled theta component F_theta(theta,phi)
% Fphi:     I'xJ' sampled phi component F_phi(theta,phi)
%           I' = 2L'+1, J = L'+1
%
% dependencies: Plm, Plmp,

J = length(muj);
I = 2*(J-1) + 1;
if nargin <= 5
    plm = Plm(L,muj); % includes monopole
    dplm = Plmp(L,muj); % includes monopole
elseif nargin == 7 && ~exist('plm') && ~exist('dplm')
    disp('bad inputs for precomputation')
    return
end

Fth = zeros(I,J);
Fphi = zeros(I,J);

sq = sqrt(1-muj.^2)';
for m = -L:L,
    if m >= 0
        ind2 = m + 1;
    else
        ind2 = I + m + 1;
    end
    if m == 0
        lowerlim = 1;
    else
        lowerlim = abs(m);
    end
    for l = lowerlim:L,
        if nargin >= 5 && strcmp(normstr,'norm')
            const = 1/sqrt(l*(l+1));
        else
            const = 1;
        end
        ind = l^2 + l + m;
        c1 = -sq.*dplm(ind+1,:);
        c2 = 1i*m*plm(ind+1,:)./sq;
        Fth(ind2,:)   = Fth(ind2,:) + const*(blm(ind)*c1 + clm(ind)*c2);
        Fphi(ind2,:)  = Fphi(ind2,:) + const*(blm(ind)*c2 - clm(ind)*c1);
    end
end

Fth = I/sqrt(2*pi)*ifft(Fth,[],1);
Fphi = I/sqrt(2*pi)*ifft(Fphi,[],1);
```

8.7.4 Vector Spherical Filter

The routine `vsfilt` is a straight forward implementation of the vector spherical filter. It works like the scalar spherical filter, `ssfilt`, to accomplish vector spherical interpolation or filtering by zero padding or truncating the expansion coefficients. It takes as input the maximum degrees of the harmonic content L and K, where $L \leq K$ on either side of the transforms. However, the sampling can be greater than the requested degree of harmonic when interpolating or filtering as long as $L \leq L'$ and $K \leq K'$. For interpolation, the input functions are $F_\theta(\theta, \phi)$ and $F_\phi(\theta, \phi)$, which are both sized $I \times J = 2L' + 1 \times L' + 1$ on a meshgrid. It returns $F_\theta(\theta', \phi')$ and $F_\phi(\theta', \phi')$, which are both sized $P \times Q = 2K' + 1 \times K' + 1$. Visa-versa for filtering.

The routine decides to interpolate or filter based on the size of the input functions and lengths of μ_j and μ_k. This routine calls `vst` and `ivst` sequentially, which means that the spherical harmonics normalization does not matter, so the default is to use partially normalized vector spherical harmonics.

```
function [Fth2 Fphi2] = vsfilt(Fth,Fphi,L,muj,wj,K,muk,wk)
% Vector spherical filter (interpolation or filtering)
% Assumes L <= K
%
% L:    Maximum degree L<=L' to interpolate up from, or filter down to
% muj:  Quadrature nodes for L'+1 points
% wj:   Quadrature weights for L'+1 ponts
% K:    Maximum degree K<=K' to interpolate up to, or filter down from
% muk:  Quadrature nodes for K'+1 points
% wk:   Quadrature weights for K'+1 points
%
% Interpolation
% Fth, Fphi:     input fields f(theta_j,phi_j):  (2L'+1)x(L'+1)
% Fth2, Fphi2:   output fields f(theta_k,phi_k): (2K'+1)x(K'+1)
%
% Filter
% Fth, Fphi:     input scalar field f(theta_k,phi_k):  (2K'+1)x(K'+1)
% Fth2, Fphi2:   output scalar field f(theta_j,phi_j): (2L'+1)x(L'+1)
%
% Dependences: vst, ivst

if L > K
    error('error, L > K')
end
if L == K
    Fth2 = Fth;
    Fphi2 = Fphi;
    return
end
[M N] = size(Fth);
J = length(muj);
I = 2*(J-1) + 1;
Q = length(muk);
P = 2*(Q-1) + 1;
if M == I % interpolation mode
    if N ~= J
        error('bad array size')
    end
    filt = 0;
elseif M == P; % filter mode
    if N ~= Q
        error('bad array size')
    end
    filt = 1;
else
    error('bad array size')
end
tot = L^2 + 2*L;
if ~filt % interpolation
    [blm clm] = vst(Fth,Fphi,L,muj,wj);
    tot2 = K^2 + 2*K;
    blm2 = zeros(tot2,1);
    clm2 = zeros(tot2,1);
    blm2(1:tot) = blm;
    clm2(1:tot) = clm;
    [Fth2 Fphi2] = ivst(blm2,clm2,K,muk);
else % filter
    [blm clm] = vst(Fth,Fphi,K,muk,wk);
    blm2 = blm(1:tot);
    clm2 = clm(1:tot);
    [Fth2 Fphi2] = ivst(blm2,clm2,L,muj);
end
```

8.7.5 Fast Vector Spherical Filter

Like in the scalar spherical filter, the vector spherical transforms above are bogged down by the Legendre transforms. There are two methods for accelerating the computation.

The first method is similar to the fast scalar spherical filter where the forward vector transform is substituted into the inverse vector transform. This results in sums of mixed products of Legendre polynomial and Legendre polynomial derivatives that look like they should be simplified with Christoffel-Darboux formulas, but expressions for simplifying the mixed terms have not been found to the best of our knowledge. This means that the 1D FMM speed up is not available. However, the sums can be precomputed, then the computation carried out with matrix-vector multiplication will be easy to implement and pretty fast.

The second method is the one that is recommended throughout the literature. Interpolation/filtering is accomplished by applying the fast scalar filter to each scalar field component of the vector field. The complication comes from the fact that the vector spherical harmonics contain derivatives of the Legendre polynomial. As a result, correction terms are needed for the harmonics at the edge of the spectrum of the field that is being interpolated or filtered. Finally, we find that method 1 and method 2 agree to machine precision with the previous routines.

Method 1 - Precomputed Matrix-Vector Multiply

Similar to the fast scalar spherical filter, we can derive a fast vector interpolation and filter procedure by substituting the forward vector transform into inverse vector transform. Defining the following terms

$$a_j = -\sqrt{1-\mu_j^2} \tag{8.94}$$

$$b_j = \frac{-i}{\sqrt{1-\mu_j^2}} = \frac{i}{a_j} \tag{8.95}$$

then (8.89) and (8.90) can be written

$$b_{lm} = \frac{1}{l(l+1)} \sum_{j=1}^{J} \left(a_j \frac{\partial \widetilde{P}_l^m(\mu_j)}{\partial \mu} f_{\theta,m}(\theta_j) + m b_j \widetilde{P}_l^m(\mu_j) f_{\phi,m}(\theta_j) \right) w_j \tag{8.96}$$

$$c_{lm} = \frac{1}{l(l+1)} \sum_{j=1}^{J} \left(m b_j \widetilde{P}_l^m(\mu_j) f_{\theta,m}(\theta_j) + (-a_j) \frac{\partial \widetilde{P}_l^m(\mu_j)}{\partial \mu} f_{\phi,m}(\theta_j) \right) w_j \tag{8.97}$$

Similar to the fast scalar operation, the sum over l in the inversion transform only goes up to a maximum harmonic L. If we are interpolating, this is the maximum degree harmonic of the coarsely sampled field (all coefficients b_{lm} and c_{lm} greater than L are zero). When filtering, the harmonic coefficients are truncated to harmonics L. In both cases, the limit of the sum is the same, all that changes is the coarse/fine sampling of either field. Letting the θ samples of the resultant field be indexed by k, equations (8.91) and (8.92) are first written more compactly as

$$f_{\theta,m}(\theta_k) = \sum_{l=|m|}^{L} b_{lm} a_k \frac{\partial \widetilde{P}_l^m(\mu_k)}{\partial \mu} + c_{lm} m (-b_k) \widetilde{P}_l^m(\mu_k) \tag{8.98}$$

$$f_{\phi,m}(\theta_k) = \sum_{l=|m|}^{L} b_{lm} m (-b_k) \widetilde{P}_l^m(\mu_k) + c_{lm} (-a_k) \frac{\partial \widetilde{P}_l^m(\mu_k)}{\partial \mu} \tag{8.99}$$

After substituting (8.96) and (8.97) into (8.98) and (8.99), exchanging the order of summation, and collecting terms we can write the combined Legendre transforms as

$$f_{\theta,m}(\theta_k) = \sum_{j=1}^{J} f_{\theta,m}(\theta_j) A_m(\mu_j, \mu_k) + f_{\phi,m}(\theta_j) B_m(\mu_j, \mu_k) \tag{8.100}$$

$$f_{\phi,m}(\theta_k) = \sum_{j=1}^{J} -f_{\theta,m}(\theta_j) B_m(\mu_j, \mu_k) + f_{\phi,m}(\theta_j) A_m(\mu_j, \mu_k) \tag{8.101}$$

where

$$A_m(\mu_j,\mu_k) = w_j a_j a_k M_{1,m}(\mu_j,\mu_k) - w_j b_j b_k m^2 M_{2,m}(\mu_j,\mu_k) \quad (8.102)$$
$$B_m(\mu_j,\mu_k) = w_j b_j a_k m M_{3,m}(\mu_j,\mu_k) + w_j a_j b_k m M_{4,m}(\mu_j,\mu_k) \quad (8.103)$$

and

$$M_{1,m}(\mu_j,\mu_k) = \sum_{l=|m|}^{L} \frac{1}{l(l+1)} \frac{\partial \widetilde{P}_l^m(\mu_j)}{\partial \mu} \frac{\partial \widetilde{P}_l^m(\mu_k)}{\partial \mu} \quad (8.104)$$

$$M_{2,m}(\mu_j,\mu_k) = \sum_{l=|m|}^{L} \frac{1}{l(l+1)} \widetilde{P}_l^m(\mu_j) \widetilde{P}_l^m(\mu_k) \quad (8.105)$$

$$M_{3,m}(\mu_j,\mu_k) = \sum_{l=|m|}^{L} \frac{1}{l(l+1)} \widetilde{P}_l^m(\mu_j) \frac{\partial \widetilde{P}_l^m(\mu_k)}{\partial \mu} \quad (8.106)$$

$$M_{4,m}(\mu_j,\mu_k) = \sum_{l=|m|}^{L} \frac{1}{l(l+1)} \frac{\partial \widetilde{P}_l^m(\mu_j)}{\partial \mu} \widetilde{P}_l^m(\mu_k) \quad (8.107)$$

In the fast scalar operator, the Christoffel-Darboux formula was used to simplify the sums over l and yield an expression that can be accelerated with the 1D FMM. Similar formulas for the above expressions have not been found. Regardless, the matrices $A_m(\mu_j,\mu_k)$, $B_m(\mu_j,\mu_k)$ can be precomputed and (8.100) and (8.101) can be computed as matrix-vector multiplication. After which, $f_{\theta,m}(\theta_k)$ and $f_{\phi,m}(\theta_k)$ are computed and then the inverse Fourier transform over m completes the filter.

The routine `fvsfilt1` implements the fast vector spherical filter using the matrix-multiplication method above. It detects whether to interpolate or filter based on the size of the input fields, which need to be sampled at the nodes of quadrature consistent with L and K. It uses `fvsfilt1AmBm` to precompute the matrices $A_m(\mu_j,\mu_k)$, $B_m(\mu_j,\mu_k)$, which take as input just L and K, where one is either interpolating from L harmonics to K, of filtering from K harmonics down to L. Use string switch `'interp'` or `'filter'` for interpolation or filter. A handy trick is that the same basic computation of the matrices applies no matter if one is interpolating or filtering, one simply swaps the sample points and nodes and weights of quadrature. The indexing then also needs to swap. The intermediate sums always only go to L, which is again the equivalent of zero padding the spherical harmonic expansion coefficients when interpolating or truncating when filtering. The routine returns the same result as `vsfilt` to machine precision. With precomputation, it is faster than `vsfilt` and becomes progressively faster as the number of harmonics increases.

```
function [Fth2 Fphi2] = fvsfilt1(Fth,Fphi,L,K,Am,Bm)
% Fast vector spherical filter (interpolation or filtering)
% Based on precomputed matrix-vector multipy.
% Assumes L <= K. Auto-detects filter or interplation based on size of Fth,
% Fphi. Am, Bm are computed with fvsfilt1AmBm for filter or interpolation.
%
% L:        Lower degree harmonic (to interpolate up from, or filter down to)
% K:        Upper degree harmonic (to interpolate up to, or filter down from)
% Am,Bm:    [JxQxI] matrix when interpolating from L to K
%           [QxJxP] matrix when filtering from K to L
%
% Interpolation
% Fth, Fphi:     input fields f(theta_j,phi_j):  (2L+1)x(L+1)
% Fth2, Fphi2:   output fields f(theta_k,phi_k): (2K+1)x(K+1)
%
% Filter
% Fth, Fphi:     input scalar field f(theta_k,phi_k):  (2K+1)x(K+1)
% Fth2, Fphi2:   output scalar field f(theta_j,phi_j): (2L+1)x(L+1)
%
% Dependences: fvsfilt1AmBm

if L > K, error('error, L > K'), end
```

```
if L == K
    Fth2 = Fth;
    Fphi2 = Fphi;
    return
end
[M N] = size(Fth);
I = 2*L+1;
J = L+1;
P = 2*K+1;
Q = K+1;
if M == I % interpolation
    if N ~= J, error('bad array size'), end
    filt = 0;
    Fth2 = zeros(P,Q);
    Fphi2 = zeros(P,Q);
    const = P/I;
elseif M == P; % filter
    if N ~= Q, error('bad array size'), end
    filt = 1;
    Fth2 = zeros(I,J);
    Fphi2 = zeros(I,J);
    const = I/P;
else
    error('bad array size')
end

% compute fft in phi
fthm = const*fft(Fth,[],1);
fphim = const*fft(Fphi,[],1);
% loop over m at L
for m = -L:L,
    if m >= 0
        ind1 = m + 1;
        ind2 = m + 1;
    else
        ind1 = I + m + 1;
        ind2 = P + m + 1;
    end
    % perform matrix vector multiply for each m
    if filt % filtering
        Fth2(ind1,:)  = fthm(ind2,:)*Am(:,:,ind1) + fphim(ind2,:)*Bm(:,:,ind1);
        Fphi2(ind1,:) = -fthm(ind2,:)*Bm(:,:,ind1) + fphim(ind2,:)*Am(:,:,ind1);
    else % interpolating
        Fth2(ind2,:)  = fthm(ind1,:)*Am(:,:,ind1) + fphim(ind1,:)*Bm(:,:,ind1);
        Fphi2(ind2,:) = -fthm(ind1,:)*Bm(:,:,ind1) + fphim(ind1,:)*Am(:,:,ind1);
    end
end
% take inverse fft in phi
Fth2 = ifft(Fth2,[],1);
Fphi2 = ifft(Fphi2,[],1);
end
```

```
function [Am, Bm] = fvsfilt1AmBm(L,K,str)
% Preperatory function for fvsfilt1, which computes the matricies
% needed for the fast vector filter. Number of points are set by quadrature
% [I,J] = [2L+1,L+1], and [P,Q] = [2K+1,K+1]
%
% L:        Lower degree harmonic (to interpolate up from, or filter down to)
% K:        Upper degree harmonic (to interpolate up to, or filter down from)
% str:      'interp' for interpolation from degree L to K
%           'filter' for filtering from degree K to L
%
% Am,Bm:    [JxQxI] matrix when interpolating from L to K
%           [QxJxI] matrix when filtering from K to L
%
% Dependencies: computeAmBm

if L > K, error('error, L > K'), end
I = 2*L+1;
```

```
J = L+1;
P = 2*K+1;
Q = K+1;
[muj wj] = legpts(J);
[muk wk] = legpts(Q);
if strcmp(str,'interp') % interp, call computeAmBm as coded
    [Am Bm] = computeAmBm(L,J,Q,I,muj,wj,muk);
elseif strcmp(str,'filter') % filter, call computeAmBm with j/k swapped
    [Am Bm] = computeAmBm(L,Q,J,I,muk,wk,muj);
else
    error('error, bad string')
end
end

function [Am Bm] = computeAmBm(L,J,Q,I,muj,wj,muk)
% Computed matricies Am Bm for parameters passed from fvsfiltmatprep
    muj = muj(:);
    wj = wj(:);
    muk = muk(:);
    aj = -sqrt(1-muj.^2);
    ak = -sqrt(1-muk.^2);
    bj = 1i./aj;
    bk = 1i./ak;
    Pj = Plm(L,muj); % includes monopole
    Ppj = Plmp(L,muj);
    Pk = Plm(L,muk);
    Ppk = Plmp(L,muk);

    % compute blocks
    M1m = zeros(J,Q,I);
    M2m = M1m;
    M3m = M1m;
    M4m = M1m;
    for m = -L:L,
        if m >= 0
            ind2 = m + 1;
        else
            ind2 = I + m + 1;
        end
        if m == 0
            lowerlim = 1;
        else
            lowerlim = abs(m);
        end
        for l = lowerlim:L,
            ind = l^2 + l + m + 1; % monopole indexing
            const = 1/(l*(l+1));
            M1m(:,:,ind2) = M1m(:,:,ind2) + const*((Ppj(ind,:)).')*(Ppk(ind,:));
            M2m(:,:,ind2) = M2m(:,:,ind2) + const*((Pj(ind,:)).')*(Pk(ind,:));
            M3m(:,:,ind2) = M3m(:,:,ind2) + const*((Pj(ind,:)).')*(Ppk(ind,:));
            M4m(:,:,ind2) = M4m(:,:,ind2) + const*((Ppj(ind,:)).')*(Pk(ind,:));
        end
    end
    Am = zeros(J,Q,I);
    Bm = Am;
    for m = -L:L,
        if m >= 0
            ind2 = m + 1;
        else
            ind2 = I + m + 1;
        end
        Am(:,:,ind2) = ((wj.*aj)*(ak.')).*M1m(:,:,ind2) - m^2*((wj.*bj)*(bk.')).*M2m(:,:,ind2);
        Bm(:,:,ind2) = m*((wj.*bj)*(ak.')).*M3m(:,:,ind2) + m*((wj.*aj)*(bk.')).*M4m(:,:,ind2);
    end
end
```

Method 2 - Fast Scalar Filter with Correction Terms

The fast scalar filter cannot simply be applied to each scalar component of the vector fields. This is because the vector spherical harmonics contain derivatives of the Legendre polynomials, which are themselves composed of Legendre polynomials at harmonic degrees one above and one below the harmonic degree of the derivative. The fast scalar filter meanwhile a) only operates on spherical harmonics that contain non-differentiated Legendre polynomials, and b) only operates up to the highest degree in the spectrum and no more. If we want to filter the scalar components of the vector field to degree L, the fast scalar filter will only be accurate for harmonic degrees less than or equal to $L-1$. There is still a way to use the fast scalar filter up to degree L, but correction terms are needed to account for the Legendre derivatives that straddle the harmonic cutoff.

The approach is to rewrite the expressions for the vector spherical harmonic expansions in terms of purely scalar spherical harmonics. This results in a handful of leftover terms which are collected to create the needed correction terms. The correction terms in [36] appear to have errors, and those given in [45] are not for normalized Legendre polynomials, so we rederive the correction terms here.

In a few places in the literature it is stated that the correction terms are only needed when filtering. The reasoning goes that when a field is interpolated there is no harmonic content above degree L, so the correction terms are not needed. However, when the Legendre derivatives are split into pure Legendre polynomials, the harmonics straddle the band edge regardless of whether a field is interpolated or filtered. We found that the correction terms are still required when interpolating in order to give the same results as our previous vector spherical filter routines.

Legendre Derivative Relations: The first step is to express the derivative of the Legendre polynomial (the $d/d\theta$ version) as a linear combination of Legendre polynomials. Start with the following two identities for unnormalized Legendre polynomials:

$$(2l+1)\cos\theta P_l^m(\cos\theta) = (l+m)P_{l-1}^m(\cos\theta) + (l-m+1)P_{l+1}^m(\cos\theta) \tag{8.108}$$

$$\sin\theta\frac{dP_l^m(\cos\theta)}{d\theta} = l\cos\theta P_l^m(\cos\theta) - (l+m)P_{l-1}^m(\cos\theta) \tag{8.109}$$

Substituting (8.108) into (8.109) it can be shown that

$$\sin\theta\frac{dP_l^m(\cos\theta)}{d\theta} = \frac{l(l-m+1)}{2l+1}P_{l+1}^m(\cos\theta) - \frac{(l+m)(l+1)}{2l+1}P_{l-1}^m(\cos\theta) \tag{8.110}$$

Multiply both sides by the normalization factor of the normalized Legendre polynomials

$$\begin{aligned}\sqrt{(l+1/2)\frac{(l-m)!}{(l+m)!}}\sin\theta\frac{dP_l^m(\cos\theta)}{d\theta} &= \sqrt{(l+1/2)\frac{(l-m)!}{(l+m)!}}\frac{l(l-m+1)}{2l+1}P_{l+1}^m(\cos\theta) \\ &\quad -\sqrt{(l+1/2)\frac{(l-m)!}{(l+m)!}}\frac{(l+m)(l+1)}{2l+1}P_{l-1}^m(\cos\theta)\end{aligned} \tag{8.111}$$

Finally, multiply the $l+1$ and $l-1$ polynomials by appropriate factors in order to apply the definition of the normalized Legendre polynomials, then simplify to get

$$\begin{aligned}\sin\theta\frac{d\widetilde{P}_l^m(\cos\theta)}{d\theta} &= \sqrt{\frac{(l+1/2)(l+1+m)}{(l+3/2)(l+1-m)}}\frac{l(l-m+1)}{2l+1}\widetilde{P}_{l+1}^m(\cos\theta) \\ &\quad -\sqrt{\frac{(l+1/2)(l-m)}{(l-1/2)(l+m)}}\frac{(l+m)(l+1)}{2l+1}\widetilde{P}_{l-1}^m(\cos\theta)\end{aligned} \tag{8.112}$$

$F_\theta(\theta,\phi)$ **Component:** Consider the θ component, (8.72), after multiplication by $\sin\theta$.

$$\sin\theta F_\theta(\theta,\phi) = \sum_{l=1}^{L}\sum_{m=-l}^{l} b_{lm}\sin\theta\frac{d}{d\theta}Y_{lm}(\theta,\phi) + c_{lm}(im)Y_{lm}(\theta,\phi) \tag{8.113}$$

Substituting (8.112), this can be written in terms of pure spherical harmonics as

$$\begin{aligned}\sin\theta F_\theta(\theta,\phi) &= \sum_{l=1}^{L}\sum_{m=-l}^{l} b_{lm}h_1(l,m)Y_{l+1,m}(\theta,\phi)\\ &\quad -\sum_{l=1}^{L}\sum_{m=-l}^{l} b_{lm}h_2(l,m)Y_{l-1,m}(\theta,\phi)\\ &\quad +\sum_{l=1}^{L}\sum_{m=-l}^{l} c_{lm}h_3(l,m)Y_{l,m}(\theta,\phi)\end{aligned} \tag{8.114}$$

where

$$h_1(l,m) = \sqrt{\frac{(l+1/2)(l+1+m)}{(l+3/2)(l+1-m)}}\frac{l(l-m+1)}{2l+1} \tag{8.115}$$

$$h_2(l,m) = \sqrt{\frac{(l+1/2)(l-m)}{(l-1/2)(l+m)}}\frac{(l+m)(l+1)}{2l+1} \tag{8.116}$$

$$h_3(l,m) = im \tag{8.117}$$

Next, let L be the highest harmonic we are interpolating from (up to K) or the highest harmonic we are filtering to (down from K) to create the scalar function $\widetilde{F}_\theta(\theta',\phi')$ at the new sampling. Then for each sum in turn, make the following substitutions respectively: $l+1\to l$, $l-1\to l$, and $l\to l$ so that the spherical harmonics have the same indices

$$\begin{aligned}\sin\theta'\widetilde{F}_\theta(\theta',\phi') &= \sum_{l=2}^{L+1}\sum_{m=-(l-1)}^{(l-1)} \widetilde{b}_{l-1,m}h_1(l-1,m)Y_{lm}(\theta',\phi')\\ &\quad -\sum_{l=0}^{L-1}\sum_{m=-(l+1)}^{(l+1)} \widetilde{b}_{l+1,m}h_2(l+1,m)Y_{lm}(\theta',\phi')\\ &\quad +\sum_{l=1}^{L}\sum_{m=-l}^{l} \widetilde{c}_{lm}h_3(l,m)Y_{lm}(\theta',\phi')\end{aligned} \tag{8.118}$$

Spin out the terms L and $L+1$ and collect the sums over l (the same expression in [36] does not appear to be correct, while the one in [45] does)

$$\begin{aligned}\sin\theta'\widetilde{F}_\theta(\theta',\phi') &= \sum_{l=0}^{L-1}\sum_{m=-l}^{l} \widetilde{d}_{l,m}Y_{lm}(\theta',\phi')\\ &\quad +\sum_{m=-L}^{L} \widetilde{e}_{L,m}Y_{Lm}(\theta',\phi')\\ &\quad +\sum_{m=-L}^{L} \widetilde{e}_{L+1,m}Y_{L+1,m}(\theta',\phi')\end{aligned} \tag{8.119}$$

where

$$\widetilde{d}_{l,m} = \widetilde{b}_{l-1,m}h_1(l-1,m) - \widetilde{b}_{l+1,m}h_2(l+1,m) + \widetilde{c}_{l,m}h_3(l,m) \tag{8.120}$$

$$\widetilde{e}_{L,m} = \widetilde{b}_{L-1,m}h_1(L-1,m) + \widetilde{c}_{L,m}h_3(L,m) \tag{8.121}$$

$$\widetilde{e}_{L+1,m} = \widetilde{b}_{L,m}h_1(L,m) \tag{8.122}$$

$F_\phi(\theta,\phi)$ Component: The ϕ component, (8.73), after multiplication by $\sin\theta$, is

$$\sin\theta F_\phi(\theta,\phi) = \sum_{l=1}^{L}\sum_{m=-l}^{l} b_{lm} im Y_{lm}(\theta,\phi) - c_{lm}\sin\theta\frac{d}{d\theta}Y_{lm}(\theta,\phi) \tag{8.123}$$

This is structurally similar to the θ component. Making the change $b_{lm} \to -c_{lm}$ and $c_{lm} \to b_{lm}$ in the above derivation we can immediately write

$$\begin{aligned}\sin\theta'\widetilde{F}_\phi(\theta',\phi') &= \sum_{l=0}^{L-1}\sum_{m=-l}^{l}\widetilde{f}_{l,m}Y_{lm}(\theta',\phi') \\ &+ \sum_{m=-L}^{L}\widetilde{g}_{L,m}Y_{Lm}(\theta',\phi') \\ &+ \sum_{m=-L}^{L}\widetilde{g}_{L+1,m}Y_{L+1,m}(\theta',\phi')\end{aligned} \tag{8.124}$$

where

$$\widetilde{f}_{l,m} = -\widetilde{c}_{l-1,m}h_1(l-1,m) + \widetilde{c}_{l+1,m}h_2(l+1,m) + \widetilde{b}_{l,m}h_3(l,m) \tag{8.125}$$

$$\widetilde{g}_{L,m} = -\widetilde{c}_{L-1,m}h_1(L-1,m) + \widetilde{b}_{L,m}h_3(L,m) \tag{8.126}$$

$$\widetilde{g}_{L+1,m} = -\widetilde{c}_{L,m}h_1(L,m) \tag{8.127}$$

(a minus sign is missing in [36])

Summary: Despite the complications, the manipulations have so far been exact. This implies that the first summation is the result obtained by applying the scalar filter directly to the vector field component up to degree $L-1$. The second and third sums correct the effects of the Legendre polynomials derivatives at the highest harmonic of the truncation. Thus the fast scalar filter can be applied to obtain the field contribution from harmonics $l = 1,..,L-1$, while the correction terms at L and $L+1$ are summed directly. The coefficients $\widetilde{d}_{l,m}$ and $\widetilde{f}_{l,m}$ are never actually computed, and neither is $h_2(l,m)$.

In [36] it is stated that the signal being filtered must be sampled on a grid one degree higher, because the field actually contains information at $L+1$, so that the number of (θ,ϕ) evaluation points needs to correspond to degree $L+1$. However, we found this sampling requirement not be the case. Rather, the scalar filter can be applied up to degree $L-1$ on a grid sampled for L. The scalar filter could also be applied up to degree L, then the correction terms need to occur at $L+1$ and $L+2$.

Note that the scalar field components are first multiplied by $\sin\theta$ before applying the fast scalar filter, then the filtered result is divided by $\sin\theta'$. The correction terms are simply divided by $\sin\theta'$ before being summed. We never divide by zero, because the Gaussian nodes never sample the poles.

Routine: The routine `fvsfilt2` is a non-optimized implementation of the algorithm above, and written only to show that these equations work. The inputs and outputs are the same as `vsfilt`. The routine calls the fast scalar filter routine `fssfilt` to interpolate from, or filter to, harmonics at $L-1$ (at the time of this writing, that routine was not optimized). Next, the routine `vst` is used to compute all the vector spherical harmonic expansion coefficients up to L of the input fields, even though only degrees $L-1$ and L are needed. It then computes and applies the correction terms, and sums the spherical harmonics at L and $L+1$ directly which are computed from `sphericalY`. The results match `vsfilt` and `fvsfilt1` with an accuracy slightly less than machine precision.

This implementation is inefficient because each of the subroutines compute all of the Legendre polynomials anew at each call. However, the fast scalar filter and the correction terms only need Legendre polynomials at degrees $L-2$, $L-1$, L, and $L+1$. The proper way to implement this is to precompute the Legendre polynomials and derivatives for these harmonics, which are then used for in-line implementations of the fast scalar filter and the combined forward and inverse Legendre transforms.

```
function [Fth2 Fphi2] = fvsfilt2(Fth,Fphi,L,muj,wj,K,muk,wk)
% Fast vector spherical filter (interpolation or filtering)
% Based on fast scalar filter with correction terms.
% Assumes L <= K. Auto-detects filter or interplation based on size of Fth,
% Fphi.
%
% L:        Lower degree harmonic (to interpolate up from, or filter down to)
% K:        Upper degree harmonic (to interpolate up to, or filter down from)
%
% Interpolation
% Fth, Fphi:     input fields f(theta_j,phi_j):  (2L+1)x(L+1)
% Fth2, Fphi2:   output fields f(theta_k,phi_k): (2K+1)x(K+1)
%
% Filter
% Fth, Fphi:     input scalar field f(theta_k,phi_k):  (2K+1)x(K+1)
% Fth2, Fphi2:   output scalar field f(theta_j,phi_j): (2L+1)x(L+1)
%
% Dependences: fssfilt, vst, sphericalY, correctionTerms, h1

if L > K, error('error, L > K'), end
if L == K
    Fth2 = Fth;
    Fphi2 = Fphi;
    return
end
[M N] = size(Fth);
J = length(muj);
I = 2*(J-1) + 1;
Q = length(muk);
P = 2*(Q-1) + 1;
if M == I % interpolation
    if N ~= J, error('bad array size'), end
    filt = 0;
    Fth2 = zeros(P,Q);
    Fphi2 = zeros(P,Q);
    mu1 = muj;
    w1 = wj;
    mu2 = muk;
    w2 = wk;
    I1 = I;
    I2 = P;
    J1 = J;
    J2 = Q;
elseif M == P; % filter
    if N ~= Q, error('bad array size'), end
    filt = 1;
    Fth2 = zeros(I,J);
    Fphi2 = zeros(I,J);
        mu1 = muk;
    w1 = wk;
    mu2 = muj;
    w2 = wj;
    I1 = P;
    I2 = I;
    J1 = Q;
    J2 = J;
else
    error('bad array size')
end
sintheta1 = repmat(sqrt(1-(mu1(:)').^2),I1,1);
sintheta2 = repmat(sqrt(1-(mu2(:)').^2),I2,1);
phi2 = 2*pi*(0:(I2-1))/I2;
th2 = acos(mu2);
[Th Phi] = meshgrid(th2,phi2);

% fast scalar filter at L-1, fields are multiplied by sin of the current
% sampling
Fth_scalar = fssfilt(Fth.*sintheta1,L-1,muj,wj,K,muk,wk);
```

```
Fphi_scalar = fssfilt(Fphi.*sintheta1,L-1,muj,wj,K,muk,wk);

% compute expansion coefficients
[blm, clm] = vst(Fth,Fphi,L,mu1,w1);

% indecies for L-1, L, L+1
Lp = L-1;
indLm1m = Lp^2 + Lp + (-Lp:Lp)';
Lp = L;
indLm = Lp^2 + Lp + (-Lp:Lp)';
Lp = L+1;
indLp1 = Lp^2 + Lp + (-Lp:Lp)';

% grab coefficents at L-1, L
bLm1m = blm(indLm1m);
bLm = blm(indLm);
cLm1m = clm(indLm1m);
cLm = clm(indLm);

% compute correction terms
[eL, eLp1, gL, gLp1] = correctionTerms(L,bLm1m,bLm,cLm1m,cLm);

% compute scalar spherical harmonics up to L+1 (will only use L and L+1)
ylm = sphericalY(L+1,Th,Phi);
grab = [indLm; indLp1];
Fth_corr = ylm(:,grab)*[eL; eLp1];
Fphi_corr = ylm(:,grab)*[gL; gLp1];
Fth_corr = reshape(Fth_corr,I2,J2);
Fphi_corr = reshape(Fphi_corr,I2,J2);

% sum the L-1 fast scalar filtered filed and the corrected field and divide
% by sin of the new sampling
Fth2 = (Fth_scalar + Fth_corr)./sintheta2;
Fphi2 = (Fphi_scalar + Fphi_corr)./sintheta2;
end

function [eL, eLp1, gL, gLp1] = correctionTerms(L,bLm1m,bLm,cLm1m,cLm)
% correction terms for K and K+1 coefficients
    tot1 = 2*L + 1;
    eL = zeros(tot1,1);
    gL = zeros(tot1,1);

    tot2 = 2*(L+1) + 1;
    eLp1 = zeros(tot2,1);
    gLp1 = zeros(tot2,1);

    % -(L-1):(L-1)
    m = ((-(L-1)):(L-1))';
    ind = (2:(tot1-1))';
    eL(ind) = bLm1m.*h1(L-1,m) + cLm(ind).*(1i*m);
        gL(ind) = -cLm1m.*h1(L-1,m) + bLm(ind).*(1i*m);
    eLp1(ind+1) = bLm(ind).*h1(L,m);
    gLp1(ind+1) = -cLm(ind).*h1(L,m);

    % -L,L
    m = [-L, L]';
    ind = [1 tot1]';
    ind2 = [2 (tot2-1)]';
    eL(ind) = cLm(ind).*(1i*m);
    gL(ind) = bLm(ind).*(1i*m);
    eLp1(ind2) = bLm(ind).*h1(L,m);
    gLp1(ind2) = -cLm(ind).*h1(L,m);
end

% helper function
function h = h1(l,m)
    h = sqrt((l+0.5).*(l+1+m)./(l+1.5)./(l+1-m)).*l.*(l-m+1)./(2*l+1);
end
```

8.8 Scattering Matrices for the FMM

8.8.1 S-matrix and Field Multiplication using Quadrature

The scattering matrix (S-matrix) (or scattering function matrix, [10]) embeds the scattering behavior of an object as a mapping between incident and scattered plane waves of different incident and scattered directions and polarizations. In the FMM, fields are treated as expansions of plane waves where many plane waves are incident on a local region at once. The outgoing scattered field in any particular direction is the sum of scattering contributions from all incident waves. In the limit, this sum can be computed as an integral of incident directions over the unit sphere. Casting the S-matrix this way allows it to be used in the structures of the FMM.

For the FMM, we choose the orthonormal basis for the S-matrix formed by $\hat{k}$, $\hat{\theta}$, and $\hat{\phi}$, such that the incident and scattered fields are defined over two far-field patterns $\mathbf{F}(\theta_s, \phi_s)$ and $\mathbf{G}(\theta_i, \phi_i)$.

$$\mathbf{E}_i(\hat{k}_i) = \left(G_\theta(\hat{k}_i)\hat{\theta} + G_\phi(\hat{k}_i)\hat{\phi}\right) e^{i\mathbf{k}_i \cdot \mathbf{r}} \tag{8.128}$$

$$\mathbf{E}_s(\hat{k}_s) = \left(F_\theta(\hat{k}_s)\hat{\theta} + F_\phi(\hat{k}_s)\hat{\phi}\right) \frac{e^{ikr}}{r} \tag{8.129}$$

$$\begin{bmatrix} F_\theta(\hat{k}_s) \\ F_\phi(\hat{k}_s) \end{bmatrix} = \begin{bmatrix} S_{\theta\theta}(\hat{k}_s, \hat{k}_i) & S_{\theta\phi}(\hat{k}_s, \hat{k}_i) \\ S_{\phi\theta}(\hat{k}_s, \hat{k}_i) & S_{\phi\phi}(\hat{k}_s, \hat{k}_i) \end{bmatrix} \begin{bmatrix} G_\theta(\hat{k}_i) \\ G_\phi(\hat{k}_i) \end{bmatrix} \tag{8.130}$$

The matrix above maps any pair of direction/polarization to any other pair. Define the operation that transforms an incoming field pattern, $\mathbf{G}(\hat{\mathbf{k}})$, to an outgoing field pattern, $\mathbf{F}(\hat{\mathbf{k}})$, as the integral of the S-matrix over the unit sphere of incident directions

$$\mathbf{F}(\hat{\mathbf{k}}_s) = \int \overline{\mathbf{S}}(\hat{\mathbf{k}}_s, \hat{\mathbf{k}}_i) \cdot \mathbf{G}(\hat{\mathbf{k}}_i) d\Omega_{k_i} \tag{8.131}$$

or

$$\begin{bmatrix} F_\theta(\hat{\mathbf{k}}_s) \\ F_\phi(\hat{\mathbf{k}}_s) \end{bmatrix} = \int \begin{bmatrix} S_{\theta\theta}(\hat{\mathbf{k}}_s, \hat{\mathbf{k}}_i) & S_{\theta\phi}(\hat{\mathbf{k}}_s, \hat{\mathbf{k}}_i) \\ S_{\phi\theta}(\hat{\mathbf{k}}_s, \hat{\mathbf{k}}_i) & S_{\phi\phi}(\hat{\mathbf{k}}_s, \hat{\mathbf{k}}_i) \end{bmatrix} \cdot \begin{bmatrix} G_\theta(\hat{\mathbf{k}}_i) \\ G_\phi(\hat{\mathbf{k}}_i) \end{bmatrix} d\Omega_{k_i} \tag{8.132}$$

To compute this exactly, the field pattern and the S-matrix are sampled at the nodes of Gaussian quadrature on a grid that is $(2L+1) \times (L+1)$ for maximum harmonic degree L. Then (8.132) can be discretized as

$$\begin{bmatrix} F_\theta(\theta_\mu, \phi_\nu) \\ F_\phi(\theta_\mu, \phi_\nu) \end{bmatrix} = \frac{2\pi}{2L+1} \sum_{i=1}^{2L+1} \sum_{j=1}^{L+1} w_j \begin{bmatrix} S_{\theta\theta}(\theta_\mu, \phi_\nu; \theta_j, \phi_i) & S_{\theta\phi}(\theta_\mu, \phi_\nu; \theta_j, \phi_i) \\ S_{\phi\theta}(\theta_\mu, \phi_\nu; \theta_j, \phi_i) & S_{\phi\phi}(\theta_\mu, \phi_\nu; \theta_j, \phi_i) \end{bmatrix} \cdot \begin{bmatrix} G_\theta(\theta_j, \phi_i) \\ G_\phi(\theta_j, \phi_i) \end{bmatrix} \tag{8.133}$$

where the spherical angles (θ_j, ϕ_i) and (θ_μ, ϕ_ν) are the samples of quadrature. Technically, only the incident directions needed to be sampled by quadrature. As always with quadrature, the poles are never sampled, so the polarization ambiguity at the poles never occurs. Writing this in matrix form, where the 2D spherical sum is over columns of the matrix, and the weights and multiplying constants are put in a diagonal matrix,

$$\begin{bmatrix} \mathbf{F}_\theta \\ \mathbf{F}_\phi \end{bmatrix} = \begin{bmatrix} \overline{\mathbf{S}}_{\theta\theta} & \overline{\mathbf{S}}_{\theta\phi} \\ \overline{\mathbf{S}}_{\phi\theta} & \overline{\mathbf{S}}_{\phi\phi} \end{bmatrix} \begin{bmatrix} \mathbf{W} & 0 \\ 0 & \mathbf{W} \end{bmatrix} \begin{bmatrix} \mathbf{G}_\theta \\ \mathbf{G}_\phi \end{bmatrix} \tag{8.134}$$

where the elements of $\mathbf{W}$ contains copies of the weights w_j as they apply to θ_j.

The routine `compute_Smatrix_quad` applies the S-matrix to an incoming field pattern when both are sampled on the points of Gaussian quadrature. It returns the scattered field sampled the same way. The fields are sized $I \times J$ where $I = 2L+1$ and $J = L+1$ for maximum harmonic degree L (as written the routine can take any sampling I and J). The S-matrix block components are $I \times J \times I \times J$ with scattered directions in the first two dimensions and incident directions in the last two dimensions. The results match the same computation when its performed by starting with a S-matrix, converting it to a T-matrix, computing the scattering via harmonic expansions, and then converting it back to an S-matrix.

```
function [Fth, Fphi] = compute_Smatrix_quad(Gth,Gphi,Stt,Stp,Spt,Spp,I,J)
% Application of S-matrix to field pattern using quadrature
%
% Gth,Gphi:           [IxJ] incident field pattern sampled at the nodes of quadrature
% Stt,Stp,Spt,Spp:    S-matrix block matrices Theta/Phi components,
%                     Size [IxJxIxJ], sampled on the points of quadrature
%                     Scattered directions dimensions 1 and 2
%                     Incident directions dimensions 3 and 4
% I,J:                I = 2L+1, J = L+1,
%
% Fth,Fphi:           [IxJ] scattered field pattern sampled at nodes of quadrature
%
% Dependencies:       legpts

% compute quadrature weights
[muj, wj] = legpts(J);

% apply quadrature weights to incident field pattern
const = 2*pi/I;
W = repmat(wj(:)',I,1);
Gth = const*W.*Gth;
Gphi = const*W.*Gphi;

% matrix multiplication
N = I*J;
Fth = reshape(Stt,N,N)*Gth(:) + reshape(Stp,N,N)*Gphi(:);
Fphi = reshape(Spt,N,N)*Gth(:) + reshape(Spp,N,N)*Gphi(:);
Fth = reshape(Fth,I,J);
Fphi = reshape(Fphi,I,J);
```

8.8.2 S-matrix to T-matrix Transformation using Quadrature

While the S-matrix is a useful 4D structure for storing the scattering properties of a target, it 1) can be difficult and inaccurate to interpolate if the propagation directions are not highly oversampled, and 2) it can be difficult to rotate between two reference frames because the wave directions and the vector components need to be transformed. On the other hand, the transition matrix (T-matrix), which relates coefficients of the incident and scattered spherical harmonic expansions, is very easy to rotate, and enables exact interpolation at arbitrary propagation directions through field expansions.

Recall the S-matrix to T-matrix transformation (7.84)

$$\begin{bmatrix} \overline{\mathbf{T}}^{MM} & \overline{\mathbf{T}}^{MN} \\ \overline{\mathbf{T}}^{NM} & \overline{\mathbf{T}}^{NN} \end{bmatrix} = \frac{k}{4\pi} \begin{bmatrix} \overline{\mathbf{L}}_1^{-1} & 0 \\ 0 & \overline{\mathbf{L}}_2^{-1} \end{bmatrix} \begin{bmatrix} \overline{\mathbf{C}}_\theta^* & \overline{\mathbf{C}}_\phi^* \\ \overline{\mathbf{B}}_\theta^* & \overline{\mathbf{B}}_\phi^* \end{bmatrix} \begin{bmatrix} \mathbf{W} & 0 \\ 0 & \mathbf{W} \end{bmatrix} \begin{bmatrix} \overline{\mathbf{S}}_{\theta\theta} & \overline{\mathbf{S}}_{\theta\phi} \\ \overline{\mathbf{S}}_{\phi\theta} & \overline{\mathbf{S}}_{\phi\phi} \end{bmatrix} \begin{bmatrix} \mathbf{W} & 0 \\ 0 & \mathbf{W} \end{bmatrix} \begin{bmatrix} \overline{\mathbf{C}}_\theta & \overline{\mathbf{B}}_\theta \\ \overline{\mathbf{C}}_\phi & \overline{\mathbf{B}}_\phi \end{bmatrix} \begin{bmatrix} \overline{\mathbf{L}}_2 & 0 \\ 0 & \overline{\mathbf{L}}_1 \end{bmatrix} \tag{8.135}$$

Our routine `vst` will compute exactly the discretized integral of the vector spherical harmonics over the unit sphere when applied from the left to the columns of $\mathbf{S}$, when the scattered directions $\mathbf{S}$ are sampled at the nodes of Gaussian quadrature. It can also be used again to compute the integral over incident directions as a left operation on $\mathbf{S}^*$.

The routine `convert_S_to_T` computes the S-matrix to T-matrix transformation (8.135). It takes as input the four S-matrix components and returns the four components of the T-matrix. Each S-matrix component is stored on a 4D grid that is $I \times J \times I \times J$, where $I = 2L + 1$ and $J = L + 1$ are sampled according to quadrature for L harmonics. The scattered directions are dimensions 1 and 2 and the incident directions are dimensions 3 and 4. The routine returns the four components of the T-matrix up to harmonic degree L all m linearly indexed, each an $N \times N$ matrix where $N = L^2 + 2L$. The routine calls `vst` with precomputed Legendre polynomials and carries out the block matrix multiplication as a sequence of operations on the columns of the S-matrix or its transpose. The columns of the S-matrix are in fact the $I \times J$ subfields that are converted to columns of the T-matrix that are pairs of coefficients length N. This uses the fully normalized vector spherical harmonics.

```
function [Tmm, Tmn, Tnm, Tnn] = convert_S_to_T(Stt,Stp,Spt,Spp,k,I,J,L)
% S-matrix to T-matrix transformation using quadrature
%
% Stt,Stp,Spt,Spp:    S-matrix block matrices Theta/Phi components,
%                     Size [IxJxIxJ], sampled on the points of quadrature
%                     Scattered directions dimensions 1 and 2
%                     Incident directions dimensions 3 and 4
% k:                  Background wavenumber
% I,J:                I = 2L+1, J = L+1,
% L:                  Maximum degree harmonic L
%
% Tmm,Tmn,Tnm,Tnn:    T-matrix block matrices
%                     Size [NxN], N = L^2 + 2L
%
% Dependencies:       vst, lmtable, legpts, Plm, Plmp

% precompute the Legendre polynomials for the transforms
[muj, wj] = legpts(J);
plm = Plm(L,muj); % includes monopole
dplm = Plmp(L,muj); % includes monopole

% indexing
Nk = I*J;
grab1_k = 1:Nk;
grab2_k = (Nk+1):(2*Nk);
N = L^2 + 2*L;
grab1_lm = 1:N;
grab2_lm = (N+1):(2*N);

% temporary arrays for storing intermediate transform and full T-matrix
S2 = zeros(2*Nk,2*N);
Tmatrix = zeros(2*N);

%%% compute the right-hand multiplication over incident field directions
% (rows of the S-matrix) using the conjugate transpose for forward
% spherical transform

% for each scattered direction, extract the incident directions as a field
for n=1:2*Nk,
    % top half of full S-matrix
    if n <= Nk
        [indi indj] = ind2sub([I J],n);
        Fth = squeeze(Stt(indi,indj,:,:));
        Fphi = squeeze(Stp(indi,indj,:,:));
    else % bottom half of full S-matrix
        [indi indj] = ind2sub([I J],n-Nk);
        Fth = squeeze(Spt(indi,indj,:,:));
        Fphi = squeeze(Spp(indi,indj,:,:));
    end
    % compute forward transform on conjugate field
    [blm clm] = vst(conj(Fth),conj(Fphi),L,muj,wj,'norm',plm,dplm);

    % conjgate transpose to bring it back and store in temporary array
    S2(n,grab1_lm) = conj(clm.');
    S2(n,grab2_lm) = conj(blm.');
end

%%% compute the left-hand multiplication over the scattered directions
% this indexes over the harmonics of the first transform (rows of the
% temporary array)
for n=1:2*N,
    Fth = reshape(S2(grab1_k,n),I,J);
    Fphi = reshape(S2(grab2_k,n),I,J);

    % apply forward transform
        [blm clm] = vst(Fth,Fphi,L,muj,wj,'norm',plm,dplm);

    % put the transform down the columns of the Tmatrix
```

```
    Tmatrix(grab1_lm,n) = clm;
    Tmatrix(grab2_lm,n) = blm;
end

% build coefficient matricies as sparse diagonal matricies
tab = lmtable(L);
l = tab(:,1);
l1 = (1i).^(-l-1);
l2 = (1i).^(-l);
diagind = 1:(2*N);
L1invL2inv = sparse(diagind,diagind,1./[l1; l2],2*N,2*N);
L2L1 = sparse(diagind,diagind,[l2; l1],2*N,2*N);

% apply coefficient matricies
Tmatrix = (k/(4*pi))*(L1invL2inv*(Tmatrix*L2L1));

% Extract the four components
Tmm = Tmatrix(grab1_lm,grab1_lm);
Tmn = Tmatrix(grab1_lm,grab2_lm);
Tnm = Tmatrix(grab2_lm,grab1_lm);
Tnn = Tmatrix(grab2_lm,grab2_lm);
```

8.8.3 T-matrix to S-matrix Transformation using Quadrature

Recall the T-matrix to S-matrix transformation (7.82)

$$\begin{bmatrix} \overline{\mathbf{S}}_{\theta\theta} & \overline{\mathbf{S}}_{\theta\phi} \\ \overline{\mathbf{S}}_{\phi\theta} & \overline{\mathbf{S}}_{\phi\phi} \end{bmatrix} = \frac{4\pi}{k} \begin{bmatrix} \overline{\mathbf{C}}_{\theta} & \overline{\mathbf{B}}_{\theta} \\ \overline{\mathbf{C}}_{\phi} & \overline{\mathbf{B}}_{\phi} \end{bmatrix} \begin{bmatrix} \overline{\mathbf{L}}_1 & 0 \\ 0 & \overline{\mathbf{L}}_2 \end{bmatrix} \begin{bmatrix} \overline{\mathbf{T}}^{MM} & \overline{\mathbf{T}}^{MN} \\ \overline{\mathbf{T}}^{NM} & \overline{\mathbf{T}}^{NN} \end{bmatrix} \begin{bmatrix} \overline{\mathbf{L}}_2^{-1} & 0 \\ 0 & \overline{\mathbf{L}}_1^{-1} \end{bmatrix} \begin{bmatrix} \overline{\mathbf{C}}_{\theta}^{*} & \overline{\mathbf{C}}_{\phi}^{*} \\ \overline{\mathbf{B}}_{\theta}^{*} & \overline{\mathbf{B}}_{\phi}^{*} \end{bmatrix} \tag{8.136}$$

Our routine `ivst` will compute exactly the matrix multiplication over the block vector spherical harmonics when applied as a left operation to the columns of the T-matrix. We can use it again to compute the right multiplication of the conjugate operation as a left multiplication of the conjugate T-matrix, $\mathbf{T}^*$.

The routine `convert_T_to_S`, computes the four components of the S-matrix given the four components of the T-matrix that contain harmonics up to degree L all m linearly indexed. It calls `ivst` to compute (8.136) as a sequence of operations over the columns of the T-matrix or its transpose. The columns of the T-matrix are treated as pairs of expansion coefficients having $N = L^2 + 2L$ harmonics each when input to `ivst` that return field quantities sized $I \times J$ that are the new columns. When done, the S-matrix is $I \times J \times I \times J$ with scattered directions in the first two dimensions and incident directions in the last two dimensions. Applied in sequence with `convert_S_to_T` the routines will return identical results. Note, these two transformations do not require physically realistic T- or S-matrices, but an unrealistic S-matrix will be filtered into a band-limited T-matrix.

```
function [Stt, Stp, Spt, Spp] = convert_T_to_S(Tmm,Tmn,Tnm,Tnn,k,I,J,L)
% T-matrix to S-matrix transformation using quadrature
%
% Tmm,Tmn,Tnm,Tnn:    T-matrix block matrices
%                     Size [NxN], N = L^2 + 2L
% k:                  Background wavenumber
% I,J:                I = 2L+1, J = L+1, S-matrix quadratute sampling
% L:                  Maximum degree harmonic L
%
% Stt,Stp,Spt,Spp:    S-matrix block matrices Theta/Phi components,
%                     Size [IxJxIxJ], sampled on the points of quadrature
%                     Scattered directions dimensions 1 and 2
%                     Incident directions dimensions 3 and 4
%
% Dependencies:       ivst, lmtable, legpts, Plm, Plmp

% precompute the Legendre polynomials for the transforms
[muj, wj] = legpts(J);
plm = Plm(L,muj); % includes monopole
dplm = Plmp(L,muj); % includes monopole
```

```
% indexing
Nk = I*J;
grab1_k = 1:Nk;
grab2_k = (Nk+1):(2*Nk);
N = L^2 + 2*L;
grab1_lm = 1:N;
grab2_lm = (N+1):(2*N);

% temporary arrays for storing intermediate transform and full T-matrix
S2 = zeros(2*N,2*Nk);
Smatrix = zeros(2*Nk,2*Nk);

%%% compute the right-hand multiplication over incident harmonics
% (rows of the T-matrix) using the conjugate transpose for the inverse
% spherical transform for each harmonic, extract the scattered harmonics
% apply normalization if

% apply coefficients as sparse diagonal matricies
tab = lmtable(L);
l = tab(:,1);
l1 = (1i).^(-l-1);
l2 = (1i).^(-l);
diagind = 1:N;
L1 = sparse(diagind,diagind,l1,N,N);
L2 = sparse(diagind,diagind,l2,N,N);
L1inv = sparse(diagind,diagind,1./l1,N,N);
L2inv = sparse(diagind,diagind,1./l2,N,N);
Tmm = L1*Tmm*L2inv;
Tmn = L1*Tmn*L1inv;
Tnm = L2*Tnm*L2inv;
Tnn = L2*Tnn*L1inv;

for n=1:2*N,
    % top half of full T-matrix
    if n <= N
        clm = squeeze(Tmm(n,:));
        blm = squeeze(Tmn(n,:));
    else % bottom half of full T-matrix
        clm = squeeze(Tnm(n-N,:));
        blm = squeeze(Tnn(n-N,:));
    end
    % compute inverse transform on conjugate coefficients
    [Fth Fphi] = ivst(conj(blm(:)),conj(clm(:)),L,muj,'norm',plm,dplm);
    % conjugate transpose to bring it back and store in temporary array
    S2(n,grab1_k) = conj(Fth(:).');
    S2(n,grab2_k) = conj(Fphi(:).');
end

%%% compute the left-hand multiplication over the scattered harmonics
% this indexes over the incident field points of the first transform (rows of the
% temporary array)
for n=1:2*Nk,
    clm = S2(grab1_lm,n);
    blm = S2(grab2_lm,n);
    % compute inverse transform
        [Fth Fphi] = ivst(blm,clm,L,muj,'norm',plm,dplm);
    % put the scattered field values down the columns of Smatrix
    Smatrix(grab1_k,n) = Fth(:);
    Smatrix(grab2_k,n) = Fphi(:);
end

sz = [I J I J];
% Extract the four components and reshape
Stt = (4*pi/k)*reshape(Smatrix(grab1_k,grab1_k),sz);
Stp = (4*pi/k)*reshape(Smatrix(grab1_k,grab2_k),sz);
Spt = (4*pi/k)*reshape(Smatrix(grab2_k,grab1_k),sz);
Spp = (4*pi/k)*reshape(Smatrix(grab2_k,grab2_k),sz);
```

8.9 1D FMM, $1/(x-x')$

Here we give the pseudo-code and algorithm of [46] that can be used to accelerate the computation of the 1-dimensional kernel $1/(x-x')$ that appears in the fast scalar spherical transform in Section 8.6.5. Specifically, this deals with the computation of

$$f(x_j) = \sum_k^N \frac{a_k}{x_j - x_k} \tag{8.137}$$

This kernel is similar to the kernel for electrostatic potentials (e.g., $1/|x-x'|$), except with the important difference that the denominator retains its sign.

Let M be the number of observation points x_j, N be the number of source points, x_k, that have amplitudes a_k which can be real or complex. In matrix form, this requires $O(NM)$ operations to perform the matrix vector multiplication. The algorithm in [46] reduces this to $O(Np+Mp)$ where p is the number of expansion terms needed to achieve a specific accuracy.

This algorithm is based on the observation that the far-field expansions of local groups of sources under this kernel can be aggregated up a binary tree-structure, expanded about another center at fixed cost, then dissaggregated. It exploits the properties of Chebyshev polynomials on the domain $x = [-1, 1]$ in order to accomplish this procedure with pre-defined precision. Two versions of the algorithm are given in [46]; we repeat the first algorithm here; the second uses SVD to accelerate the computation.

For x_j and x_k in the range $x = [a, b]$, the inputs can be rescaled with the affine transformation

$$x' = \frac{2}{b-a}(x+a) - 1 \tag{8.138}$$

$$x = \frac{b-a}{2}(x'+1) - a \tag{8.139}$$

Using (8.139) in (8.137) yields x'_j and x'_k on $x = [-1, 1]$.

$$f(x'_j) = \frac{2}{b-a} \sum_k^N \frac{a_k}{x'_j - x'_k} \tag{8.140}$$

Therefore only the source and observation points need to be transformed using (8.138), and the scale factor $2/(b-a)$ applied to the amplitudes, the algorithm is otherwise equivalent.

Setup

- p is an integer that is the size of the Chebyshev expansions. p is the same for all expansions on $x = [-1, 1]$ and given by $p = \lceil -\log_5(\epsilon) \rceil$ where $0 < \epsilon < 1$ is the desired precision.
- $t_1, ..., t_p$ are the Chebyshev nodes of order p on $x = [-1, 1]$, given by

$$t_i = \cos\left(\frac{2i-1}{p}\frac{\pi}{2}\right) \tag{8.141}$$

- The expansion functions are given by the polynomials

$$u_j(t) = \prod_{\substack{k=1 \\ k \neq j}}^{p} \frac{t - t_k}{t_j - t_k} \tag{8.142}$$

- The far-field due to sources in $x = [x_o - r, x_o + r]$ is given by

$$f_{\text{far}}(x) = \sum_j^p \Phi_j u_j\left(\frac{3r}{x - x_o}\right) \tag{8.143}$$

$$\Phi_j = \sum_k^p a_k \frac{t_j}{3r - t_j(x_k - x_o)} \tag{8.144}$$

where Φ_k are the far-field expansion coefficients.

- The local field in $x = [y_o - r, y_o + r]$ is given by

$$f_{\text{loc}}(x) = \sum_j^p \Psi_j u_j \left(\frac{x - y_o}{r} \right) \tag{8.145}$$

$$\Psi_j = \sum_k^p \frac{a_k}{r t_j - (x_k - y_o)} \tag{8.146}$$

  where Ψ_k are the local field expansion coefficients.
- s is an integer and is the number of points in each subinterval at the finest level. It is recommended to set $s \approx 2p$.
- $nlevs = \lceil \log_2(N/s) \rceil$ is the level of finest subinterval and the total number of levels.
- $\Phi_{l,i}$ are the p-term far-field coefficients at level l, subinterval i.
- $\Psi_{l,i}$ are the p-term local-field coefficients at level l, subinterval i.
- M_L and M_R are $p \times p$ matrices that aggregate the far-field expansions of left and right subinterval (children) to a far-field expansion at the next higher level (parent). These are given by ([46, Eq. 78] has a typo)

$$M_L(i,j) = u_j \left(\frac{3t_i}{6 + t_i} \right) \tag{8.147}$$

$$M_R(i,j) = u_j \left(\frac{3t_i}{6 - t_i} \right) \tag{8.148}$$

- S_L and S_R are $p \times p$ matrices that disaggregate the local expansion of a parent to its left and right children

$$S_L(i,j) = u_j \left(\frac{t_i - 1}{2} \right) \tag{8.149}$$

$$S_R(i,j) = u_j \left(\frac{t_i + 1}{2} \right) \tag{8.150}$$

- T_1, T_2, T_3, T_4 are $p \times p$ matrices that translate the far-field expansions of the well-separated subdivisions to the local expansion of a subinterval at the same level. The far-field subdivisions are separated from the local subinterval by -3, -2, 2 and 3 positions, respectively.

$$T_1(i,j) = u_j \left(\frac{3}{t_i - 6} \right) \tag{8.151}$$

$$T_2(i,j) = u_j \left(\frac{3}{t_i - 4} \right) \tag{8.152}$$

$$T_3(i,j) = u_j \left(\frac{3}{t_i + 4} \right) \tag{8.153}$$

$$T_4(i,j) = u_j \left(\frac{3}{t_i + 6} \right) \tag{8.154}$$

Algorithm psuedo-code

1. Set the expansion size p, choose s, and compute $nlevs$. Precompute Chebyshev coefficients and translation matrices.
2. Determine the far-field expansions at the finest level.

   do $i = 1, ..., 2^{nlevs}$

   Compute p-term far-field expansions $\Phi_{nlevs,i}$ using (8.144) due to sources at x_k which lie in subinterval i at level $nlevs$.

end

3. Determine the p-term far-field expansion at each subinterval at every level by shifting and adding the far-field expansions of the subintervals children

do $l = nlevs - 1, ..., 1$

do $i = 1, ..., 2^l$

$\Phi_{l,i} = M_L \cdot \Phi_{l+1,2i-1} + M_R \cdot \Phi_{l+1,2i}$

end

end

4. Determine p-term local expansion at each subinterval at each level by 1) disaggregating the parent's local expansion, 2) adding the far-field translation to local translation of well-separated subintervals at the same level, but that have not been accounted for at the parent's level. (The equations for this step given in [46] do not work as they appear, but the following does)

do $l = 1, ..., nlevs - 1$

do $i = 1, ..., 2^l$

$\Psi_{l+1,2i-1} = S_L \cdot \Psi_{l,i} + T_2 \cdot \Phi_{l+1,2i-3} + T_3 \cdot \Phi_{l+1,2i+1} + T_4 \cdot \Phi_{l+1,2i+2}$ $\Psi_{l+1,2i} = S_R \cdot \Psi_{l,i} + T_1 \cdot \Phi_{l+1,2i-3} + T_2 \cdot \Phi_{l+1,2i-2} + T_3 \cdot \Phi_{l+1,2i+2}$

end

end

5. Evaluate the local expansion at the finest level

do $i = 1, ..., 2^{nlevs}$

Evaluate p-term local expansions $\Psi_{nlevs,i}$ using (8.145) at points x_j which lie in subinterval i at level $nlevs$.

end

6. Add the near-neighbor contributions directly

do $i = 1, ..., 2^{nlevs}$

For each point x_j in subinterval i at level $nlevs$, compute the contribution of all x_k in subintervals $i-1$, i, $i+1$ using (8.137), and add the result to the local expansion already evaluated before.

end

Routines

The 1D FMM is implemented with two routines `fmm1prep` and `fmm1`. `fmm1prep` is a preparatory function that precomputes the setup parameters of the algorithm and translation matrices using (8.141) through (8.154). It takes as inputs observation points x_j and source points x_k on $x = [-1, 1]$, source amplitudes a_k (real or complex), and precision ϵ. s is optional, the default is $s = 2p$. The outputs are M_L, M_R, S_L, S_R, T_1, T_2, T_3, T_4, etc. `fmm1` takes the outputs from `fmm1prep` and executes the algorithm given by the pseudo-code. The pair of routines is set up so that `fmm1` can be called with new source amplitudes for the same source and observation points and the same outputs of `fmm1prep`.

Both routines rely on linear indexing to bookkeep the values of variables. The total number of expansions needed, and thus total number of subdivisions on the binary tree, is

$$B = \sum_{i=1}^{nlevs} 2^i = 2^{nlevs+1} - 2 \tag{8.155}$$

The expansion coefficients vectors $\Phi_{l,i}$ and $\Psi_{l,i}$ are stored on $p \times B$ arrays and accessed with column index

$$I(l,i) = 2^l - 2 + i \tag{8.156}$$

for $l \geq 1, i = 1, ..., 2^l$. This is provided by the helper function `box2ind`, which has been replaced by inline computation in the code. The basis functions, $u_j(t)$, are provided in the routine `fmm1u`. Note, the expansions at the top-most level, $l = 1$, are never used except to initialize one of the loops with zeros.

Because Matlab's matrix vector multiplication is highly optimized, it is actually faster for small problems to precompute the elements of the matrix and let Matlab do the computation directly. This works to a point. These routines require comparably no storage and are the only path forward for very large problems.

```
function [f] = fmm1(ak,S)
% 1D fast multipole method on x = [-1 1] for kernel of the form
%
%       f(x_j) = sum_k^N a_k/(x_j - x_k)
%
% Specialized for non-overlapping source/observation points.  Takes
% inputs from the preparatory funciton fmm1prep.
%
% ak:       source amplitudes (real or complex)
% S:        precomputed structure from fmm1prep
%
% f:        function evaluted at observation points xj
%
% dependencies: fmm1prep, fmm1u, box2ind

f = zeros(S.Nj,1);
S.Phi(:) = 0;
S.Psi(:) = 0;
ak = ak(:);

% compute far-field at nlevs boxes, k points
const = 2^S.nlevs - 2;
for b = 1:length(S.xk_notempty),
    i = S.xk_notempty(b);
    ind = const + i;           % ind = 2^S.nlevs - 2 + i;  box2ind(S.nlevs,i)
    S.Phi(:,ind) = S.uk_far{i}*ak(S.xk_ind{i});
end

% far-field at each subinterval, each level
for l=(S.nlevs-1):-1:2,
    tl = 2^l;
    tlp1m3 = 2^(l+1) - 3;
   for i=1:tl,
       ind1 = tl - 2 + i;    % ind1 = 2^l - 2 + i;        box2ind(l,i)
       ind2 = tlp1m3 + 2*i;  % ind2 = 2^(l+1) + 2*i - 3;  box2ind(l+1,2*i-1)
       ind3 = ind2 + 1;      % ind3 = ind2 + 1;           box2ind(l+1,2*i)
       S.Phi(:,ind1) = S.ML*S.Phi(:,ind2) + S.MR*S.Phi(:,ind3);
   end
end

% local expansions each level, each subinterval
for l=1:(S.nlevs-1),
    tl = 2^l;
    tlp1 = 2^(l+1);
    for i=1:(2^l),
        ti = 2*i;
        tli = tlp1 + ti;
        if (ti - 3) >= 1        % (2*i-3) >= 1
            ind = tli - 5;  % ind = 2^(l+1) + 2*i - 5; box2ind(l+1,2*i-3)
            tmp1L = S.T2*S.Phi(:,ind);
            tmp1R = S.T1*S.Phi(:,ind);
        else
            tmp1L = 0;
            tmp1R = 0;
        end
        if (ti-2) >= 1      % (2*i-2) >= 1
            ind = tli - 4;  % ind = 2^(l+1) + 2*i - 4; box2ind(l+1,2*i-2);
```

```
            tmp2 = S.T2*S.Phi(:,ind);
        else
            tmp2 = 0;
        end
        if (ti+1) <= tlp1         % (2*i+1) <= 2^(l+1)
            ind = tli - 1;  % ind = 2^(l+1) + 2*i - 1; box2ind(l+1,2*i+1)
            tmp3 = S.T3*S.Phi(:,ind);
        else
            tmp3 = 0;
        end
        if (ti+2) <= tlp1        % (2*i+2) <= 2^(l+1)
            ind = tli;      % ind = 2^(l+1) + 2*i; box2ind(l+1,2*i+2)
            tmp4L = S.T4*S.Phi(:,ind);
            tmp4R = S.T3*S.Phi(:,ind);
        else
            tmp4L = 0;
            tmp4R = 0;
        end
        ind1 = tli - 3;     % ind1 = 2^(l+1) + 2*i - 3; box2ind(l+1,2*i-1)
        ind2 = tli - 2;     % ind2 = 2^(l+1) + 2*i - 2; box2ind(l+1,2*i)
        ind3 = tl + i - 2;        % ind3 = 2^l + i - 2;       box2ind(l,i)
        S.Psi(:,ind1)   = S.SL*S.Psi(:,ind3) + tmp1L + tmp3 + tmp4L;
        S.Psi(:,ind2)   = S.SR*S.Psi(:,ind3) + tmp1R + tmp2 + tmp4R;
     end
end

% evaluate local expansions at subset of xj
const = 2^S.nlevs - 2;
for b = 1:length(S.xj_notempty),
    i = S.xj_notempty(b);
    ind = const + i;        % ind = 2^S.nlevs - 2; box2ind(S.nlevs,i)
    f(S.xj_ind{i}) = S.uj_local{i}*S.Psi(:,ind);
end

% evaluate near-neighbors directly with sparse matrix-vector multiply
f = f + S.M*ak;

end



function [S] = fmm1prep(xk,xj,epsilon,s)
% Preparatory function for 1D fast multipole method routine fmm1
%
% xk:       source points, N x 1
% xj:       observation points, N x 1
% epsilon:  precision of the computation
% s:        [optional] number of points in each bin at finest level
%           default: s = 2*p
%
% Outputs stored in structure S (also ak and xj):
%
% ML,MR,SL,SR,T1,T2,T3,T4:
%           p x p translation matrices
% ti:       p x 1 Chebyshev coefficents on x = [-1 1]
% p:        maximum degree of expansions
% nlevs:    number of levels
% r:        radius of subintervals at finest level (half the width)
% cen:      box centers at finest level
% xj_ind:   box index of observation points at finest level
% xk_ind:   box index of source points at finest level
% Phi, Psi: p x B array, preallocated storage for expansion coefficients
% B:        total number of boxes

% dependencies: fmm1u

p = ceil(-log(epsilon)/log(5));
if nargin == 3
```

```
    s = 2*p;
end
xj = xj(:);
xk = xk(:);
Nj = length(xj);
Nk = length(xk);
nlevs = ceil(log2(Nk/s));
if nlevs < 1
   nlevs = 1;
end
B = 2^(nlevs+1) - 2;
Phi = zeros(p,B);
Psi = zeros(p,B);

% parameters at finest level
nbox = 2^(nlevs);
width = (1/2)^(nlevs-1);
r = width/2;
cen = (0:(nbox-1))*width - 1 + r;

% Chebyshev coefficients
ti = cos((2*(1:p)-1)*pi/p/2);

% box index at the finest level of the points
xj_ind = ceil((xj + 1)/2*nbox);
xj_ind(xj_ind == 0) = 1;
xk_ind = ceil((xk + 1)/2*nbox);
xk_ind(xk_ind == 0) = 1;

xj_ind_list = cell(nbox,1);
xk_ind_list = cell(nbox,1);
xj_notempty = [];
xk_notempty = [];
nn_ind_list = cell(nbox,1);
uj_local = cell(nbox,1);
uk_far = cell(nbox,1);
for i=1:nbox,
    xj_ind_list{i} = find(xj_ind == i);
    xk_ind_list{i} = find(xk_ind == i);
    if ~isempty(xj_ind_list{i})
        xj_notempty = [xj_notempty; i];
        uj_local{i} = fmm1u((xj(xj_ind_list{i})-cen(i))/r,ti);
    end
    if ~isempty(xk_ind_list{i})
        xk_notempty = [xk_notempty; i];
        tmp = zeros(p,length(xk_ind_list{i}));
        for pp=1:p,
            tmp(pp,:) = ti(pp)./(3*r-ti(pp)*(xk(xk_ind_list{i})-cen(i)));
        end
        uk_far{i} = tmp;
    end
    ind1 = find(xk_ind == i-1);
    ind2 = find(xk_ind == i);
    ind3 = find(xk_ind == i+1);
    indk = [ind1; ind2; ind3];
    nn_ind_list{i} = indk;
end

% compute contributions from neighbor boxes directly
% store as sparse matrix
row = [];
col = [];
elm = [];
for i=1:nbox,
    indj = xj_ind_list{i};
    indk = nn_ind_list{i};
    [indK, indJ] = ndgrid(indk,indj);
    M = 1./(xj(indJ) - xk(indK));
```

```
    M(isnan(M)) = 0;
    row = [row; indJ(:)];
    col = [col; indK(:)];
    elm = [elm; M(:)];
end
M = sparse(row,col,elm,Nj,Nk);

% M, S, T
ML = fmm1u(3*ti./(6+ti),ti);
MR = fmm1u(3*ti./(6-ti),ti);
SL = fmm1u((ti-1)/2,ti);
SR = fmm1u((ti+1)/2,ti);
T1 = fmm1u(3./(ti+6),ti);
T2 = fmm1u(3./(ti+4),ti);
T3 = fmm1u(3./(ti-4),ti);
T4 = fmm1u(3./(ti-6),ti);

% store in structure
S.xk = xk;  S.xj = xj;  S.ML = ML;  S.MR = MR; S.SL = SL;
S.SR = SR;  S.T1 = T1;  S.T2 = T2;  S.T3 = T3;
S.T4 = T4;  S.ti = ti;  S.Nk = Nk;  S.Nj = Nj;
S.p = p; S.nlevs = nlevs; S.nbox = nbox; S.r = r;
S.cen = cen; S.xj_ind = xj_ind_list; S.xk_ind = xk_ind_list;
S.nn_list = nn_ind_list; S.uj_local = uj_local;
S.uk_far = uk_far; S.Phi = Phi; S.Psi = Psi; S.B = B; S.M = M;
S.xj_notempty = xj_notempty; S.xk_notempty = xk_notempty;

end



function [u] = fmm1u(t,ti)
% Basis functions for 1D fast-multipole method
%
% u_j(t) = prod_{k=1, k~=j}^p (t-t_k)/(t_j-t_k)
%
% t:    input arguments
% ti:   p Chebychev coefficients on x = [-1 1]
%
% u:    basis function evaluated at t, size: length(t) x p

t = t(:);
p = length(ti);
n = length(t);
u = zeros(n,p);
for j=1:p
    tmp = ones(n,1);
    for k=1:p,
        if k~=j
            tmp = tmp.*((t-ti(k))/(ti(j)-ti(k)));
        end
    end
    u(:,j) = tmp;
end



function ind = box2ind(l,i)
% Column index for 1D fast multipole method expansion coefficients
%
% l,i:  level and subdivision number
%
% ind:  column index

ind = 2^l - 2 + i ;
```

Chapter 9

Kirchhoff Approximation

The Kirchhoff approximation is used to derive scattering solutions from surfaces or facets that are assumed to be locally smooth, [47]. The idea of locally smooth is usually expressed as a constraint on the minimum local radius of curvature of a rough surface relative to the wavelength, or to say that the correlation length of the surface must be many times larger than the root mean squared height. The core idea of the Kirchhoff approximation is to replace the total electrical and magnetic field solutions at a dielectric interface with Fresnel reflection, rather than to solve the full multiple scattering problem. This formulation can be used to derive the radar cross sections of simple shapes (rectangles, circular disks, etc.), or isolated facets that are used to create larger surface discretizations.

In this chapter, we 1) give the main steps of the Kirchhoff approximation based on [47], 2) give code for the Fresnel reflection coefficients, 3) give solutions for the surface phase integral over canonical facet shapes, 4) derive expressions for the S-matrix of an arbitrarily shaped surface facet, and 5) derive the equations for specular and backscatter radar cross sections of canonical shapes.

9.1 Derivation of the Kirchhoff Approximation

Let the incident field be a plane wave defined relative to the origin

$$\mathbf{E}_i = \mathbf{e}_i E_o e^{i\mathbf{k}_i \cdot \mathbf{r}} \tag{9.1}$$

where E_o is the electric field amplitude, $\mathbf{e}_i$ is the polarization vector, $\mathbf{k}_i$ is the incident wave vector, and $\mathbf{r}$ is the position vector. The reflected and transmitted fields above and below a dielectric boundary are given by (9.2) and (9.3), respectively,

$$\mathbf{E}_r(\mathbf{r}) = \int_S \left\{ i\omega\mu_o \overline{\mathbf{G}}_1(\mathbf{r},\mathbf{r}') \cdot \hat{n}' \times \mathbf{H}(\mathbf{r}') + \left[\nabla \times \overline{\mathbf{G}}_1(\mathbf{r},\mathbf{r}')\right] \cdot \hat{n}' \times \mathbf{E}(\mathbf{r}') \right\} dS' \tag{9.2}$$

$$\mathbf{E}_t(\mathbf{r}) = \int_S \left\{ i\omega\mu_o \overline{\mathbf{G}}_2(\mathbf{r},\mathbf{r}') \cdot \hat{n}'_d \times \mathbf{H}(\mathbf{r}') + \left[\nabla \times \overline{\mathbf{G}}_2(\mathbf{r},\mathbf{r}')\right] \cdot \hat{n}'_d \times \mathbf{E}(\mathbf{r}') \right\} dS' \tag{9.3}$$

where k_1 and k_2 are the wavenumbers in the upper and lower regions, respectively, $\hat{n}$ and $\hat{n}_d$ are the upward and downward pointing surface normals, and $\mathbf{E}(\mathbf{r}')$ and $\mathbf{H}(\mathbf{r}')$ are the fields on the boundary.

The dyadic Green's functions in both regions are given by:

$$\overline{\mathbf{G}}_1(\mathbf{r},\mathbf{r}') = \left[\bar{\mathbf{I}} + \frac{1}{k_1^2}\nabla\nabla\right] \frac{e^{ik_1|\mathbf{r}-\mathbf{r}'|}}{4\pi|\mathbf{r}-\mathbf{r}'|} \tag{9.4}$$

$$\overline{\mathbf{G}}_2(\mathbf{r},\mathbf{r}') = \left[\bar{\mathbf{I}} + \frac{1}{k_2^2}\nabla\nabla\right] \frac{e^{ik_2|\mathbf{r}-\mathbf{r}'|}}{4\pi|\mathbf{r}-\mathbf{r}'|} \tag{9.5}$$

In the far-field these simplify to

$$\overline{\mathbf{G}}_1(\mathbf{r},\mathbf{r}') \approx \left(\bar{\mathbf{I}} - \hat{k}_r\hat{k}_r\right) \frac{e^{ik_1 r}}{4\pi r} \exp(-i\mathbf{k}_r \cdot \mathbf{r}') \tag{9.6}$$

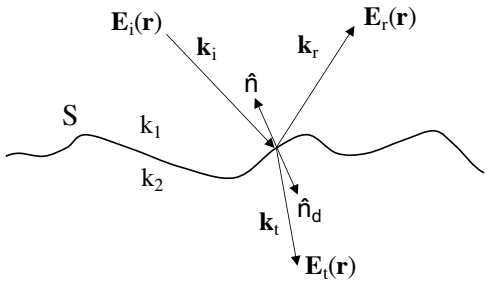


Figure 9.1: Geometry for the Kirchhoff approximation at a dielectric interface

$$\overline{\mathbf{G}}_2(\mathbf{r},\mathbf{r}') \approx \left(\overline{\mathbf{I}} - \hat{k}_t\hat{k}_t\right)\frac{e^{ik_2 r}}{4\pi r}\exp(-i\mathbf{k}_t\cdot\mathbf{r}') \tag{9.7}$$

where $\mathbf{k}_r = k_1\hat{k}_r$ and $\mathbf{k}_t = k_2\hat{k}_t$ are the reflected and transmitted wave vectors. The wave vectors are defined in the medium of propagation. Substituting these

$$\mathbf{E}_r(\mathbf{r}) = \frac{ik_1 e^{ik_1 r}}{4\pi r}\left(\overline{\mathbf{I}} - \hat{k}_r\hat{k}_r\right)\int_S \left\{\hat{k}_r\times[\hat{n}'\times\mathbf{E}(\mathbf{r}')] + \eta_1[\hat{n}'\times\mathbf{H}(\mathbf{r}')]\right\}e^{-i\mathbf{k}_r\cdot\mathbf{r}'}dS' \tag{9.8}$$

$$\mathbf{E}_t(\mathbf{r}) = \frac{ik_2 e^{ik_2 r}}{4\pi r}\left(\overline{\mathbf{I}} - \hat{k}_t\hat{k}_t\right)\int_S \left\{\hat{k}_t\times[\hat{n}'_d\times\mathbf{E}(\mathbf{r}')] + \eta_2[\hat{n}'_d\times\mathbf{H}(\mathbf{r}')]\right\}e^{-i\mathbf{k}_t\cdot\mathbf{r}'}dS' \tag{9.9}$$

Next, define the orthonormal system $(\hat{p}_i, \hat{q}_i, \hat{k}_i)$ at point $\mathbf{r}'$

$$\hat{q}_i = \frac{\hat{k}_i\times\hat{n}}{|\hat{k}_i\times\hat{n}|} \tag{9.10}$$

$$\hat{p}_i = \hat{q}_i\times\hat{k}_i \tag{9.11}$$

where $\hat{n}(\mathbf{r}') = -\hat{n}_d(\mathbf{r}')$. Under the Kirchhoff approximation, the tangential electric and magnetic files for TE and TM fields are given by (9.12) and (9.13), [47],

$$\hat{n}\times\mathbf{E}(\mathbf{r}') = E_o\left\{(\hat{e}_i\cdot\hat{q}_i)(\hat{n}\times\hat{q}_i)(1+R^{\mathrm{TE}}) + (\hat{e}_i\cdot\hat{p}_i)(\hat{n}\cdot\hat{k}_i)\hat{q}_i(1-R^{\mathrm{TM}})\right\}e^{i\mathbf{k}_i\cdot\mathbf{r}'} \tag{9.12}$$

$$\hat{n}\times\mathbf{H}(\mathbf{r}') = \frac{E_o}{\eta_1}\left\{-(\hat{e}_i\cdot\hat{q}_i)(\hat{n}\cdot\hat{k}_i)\hat{q}_i(1-R^{\mathrm{TE}}) + (\hat{e}_i\cdot\hat{p}_i)(\hat{n}\times\hat{q}_i)(1+R^{\mathrm{TM}})\right\}e^{i\mathbf{k}_i\cdot\mathbf{r}'} \tag{9.13}$$

where R^{TE} and R^{TM} are the Fresnel reflection coefficients given in Section 9.2. The local incidence angle is

$$\cos\theta_i = -\hat{n}\cdot\hat{k}_i \tag{9.14}$$

Substituting these into the surface integral equations

$$\mathbf{E}_r(\mathbf{r}) = \frac{ik_1 e^{ik_1 r}}{4\pi r}E_o\left(\overline{\mathbf{I}} - \hat{k}_r\hat{k}_r\right)\cdot\int_S \mathbf{F}(\mathbf{r}')e^{i(\mathbf{k}_i-\mathbf{k}_r)\cdot\mathbf{r}'}dS' \tag{9.15}$$

$$\mathbf{E}_t(\mathbf{r}) = -\frac{ik_2 e^{ik_2 r}}{4\pi r}E_o\left(\overline{\mathbf{I}} - \hat{k}_t\hat{k}_t\right)\cdot\int_S \mathbf{N}(\mathbf{r}')e^{i(\mathbf{k}_i-\mathbf{k}_t)\cdot\mathbf{r}'}dS' \tag{9.16}$$

where

$$\begin{aligned}\mathbf{F}(\mathbf{r}') = {} & -(\hat{e}_i\cdot\hat{q}_i)(\hat{n}\cdot\hat{k}_i)\hat{q}_i(1-R^{\mathrm{TE}}) + (\hat{e}_i\cdot\hat{p}_i)(\hat{n}\times\hat{q}_i)(1+R^{\mathrm{TM}}) \\ & +(\hat{e}_i\cdot\hat{q}_i)(\hat{k}_r\times(\hat{n}\times\hat{q}_i))(1+R^{\mathrm{TE}}) + (\hat{e}_i\cdot\hat{p}_i)(\hat{n}\cdot\hat{k}_i)(\hat{k}_r\times\hat{q}_i)(1-R^{\mathrm{TM}})\end{aligned} \tag{9.17}$$

$$\begin{aligned}\mathbf{N}(\mathbf{r}') = {} & -\frac{\eta_2}{\eta_1}(\hat{e}_i\cdot\hat{q}_i)(\hat{n}\cdot\hat{k}_i)\hat{q}_i(1-R^{\mathrm{TE}}) + \frac{\eta_2}{\eta_1}(\hat{e}_i\cdot\hat{p}_i)(\hat{n}\times\hat{q}_i)(1+R^{\mathrm{TM}}) \\ & +(\hat{e}_i\cdot\hat{q}_i)(\hat{k}_t\times(\hat{n}\times\hat{q}_i))(1+R^{\mathrm{TE}}) + (\hat{e}_i\cdot\hat{p}_i)(\hat{n}\cdot\hat{k}_i)(\hat{k}_t\times\hat{q}_i)(1-R^{\mathrm{TM}})\end{aligned} \tag{9.18}$$

F and **N** only differ in the constants and wave vectors. Due to the plane-wave approximation at the surface, the surface curvature must be large enough so that the Kirchhoff approximation is valid. Whether this is met or not, there are a number of ways to compute the integral depending on the application. One way is to finely discretize the surface at step sizes much smaller than the wavelength ($dS < \lambda/10$).

9.2 Fresnel Reflection Coefficients

The Fresnel reflection coefficients for flat interfaces are given by, [47],

$$R^{\mathrm{TE}} = \frac{\cos\theta_i - \sqrt{(\epsilon_2/\epsilon_1) - \sin^2\theta_i}}{\cos\theta_i + \sqrt{(\epsilon_2/\epsilon_1) - \sin^2\theta_i}} \tag{9.19}$$

$$R^{\mathrm{TM}} = \frac{(\epsilon_2/\epsilon_1)\cos\theta_i - \sqrt{(\epsilon_2/\epsilon_1) - \sin^2\theta_i}}{(\epsilon_2/\epsilon_1)\cos\theta_i + \sqrt{(\epsilon_2/\epsilon_1) - \sin^2\theta_i}} \tag{9.20}$$

The transmission coefficients are $T^{\mathrm{TE}} = 1 + R^{\mathrm{TE}}$ and $T^{\mathrm{TM}} = 1 + R^{\mathrm{TM}}$. In [48], there is a negative sign in the reflection coefficients that does not appear in [47]. This is due to the sign convention of incoming/outgoing field components. In [47], the incoming and outgoing fields are defined relative to the p, q coordinate projection, while in [48] they are defined relative to incoming and outgoing plane wave directions, which flips the sign of E and H fields on reflection.

The routines `fresnelTE` and `fresnelTM` compute the Fresnel reflection coefficients given the dielectric constants on either side of an interface, which can be complex, and the incidence angle in degrees.

```
function [RTE] = fresnelTE(er1,er2,theta_inc_deg)
% Fresnel reflection coefficient - TE
%
% er1, er2:         Relative permittivity (complex) in regions 1 and 2
% theta_inc_deg:    Incident angle (degrees)
%
% RTE:              TE reflection coefficient

ratio = er2./er1;
sq = sqrt(ratio-sind(theta_inc_deg).^2);
RTE = (cosd(theta_inc_deg) - sq)./(cosd(theta_inc_deg) + sq);


function [RTM] = fresnelTM(er1,er2,theta_inc_deg)
% Fresnel reflection coefficient - TM
%
% er1, er2:         Relative permittivity (complex) in regions 1 and 2
% theta_inc_deg:    Incident angle (degrees)
%
% RTM:              TM reflection coefficient

ratio = er2./er1;
sq = sqrt(ratio-sind(theta_inc_deg).^2);
RTM = (ratio.*cosd(theta_inc_deg) - sq)./(ratio.*cosd(theta_inc_deg) + sq);
```

9.3 Facetized Surfaces

One way to compute equations (9.15) and (9.16) is to break up a large surface into many flat facets and compute the surface integrals analytically over the shapes of the facets. Discretizing the surface integral as a sum over large facets we have

$$\mathbf{E}_r(\mathbf{r}) \approx \frac{ik_1 e^{ik_1 r}}{4\pi r} E_o \left(\bar{\mathbf{I}} - \hat{k}_r\hat{k}_r\right) \cdot \sum_n \mathbf{F}(\mathbf{r}_n) \int_{S_n} e^{i(\mathbf{k}_i - \mathbf{k}_r)\cdot\mathbf{r}} dS \tag{9.21}$$

$$\mathbf{E}_t(\mathbf{r}) \approx -\frac{ik_2 e^{ik_2 r}}{4\pi r} E_o \left(\bar{\mathbf{I}} - \hat{k}_t\hat{k}_t\right) \cdot \sum_n \mathbf{N}(\mathbf{r}_n) \int_{S_n} e^{i(\mathbf{k}_i - \mathbf{k}_t)\cdot\mathbf{r}} dS \tag{9.22}$$

where S_n is the surface of the nth facet. This assumes $\mathbf{F}(\mathbf{r})$ and $\mathbf{N}(\mathbf{r})$ are constant over the facet, i.e., the facet is flat and illuminated with plane waves. We are left with having to compute the surface phase integral over different facet shapes which can be written compactly in terms of the wave vector difference, $\mathbf{K}$, as

$$I = \int_S e^{i\mathbf{K}\cdot\mathbf{r}} dS \tag{9.23}$$

where $\mathbf{K} = \mathbf{k}_i - \mathbf{k}_r$ or $\mathbf{K} = \mathbf{k}_i - \mathbf{k}_t$. The surface phase integral is the 2D Fourier transform over the shape of the facet in the wave vector difference domain.

9.4 Surface Phase Integral

The surface phase integral is the 2D Fourier transform of the shape of a flat facet into the wave vector difference domain. Here we derive analytic expressions for the surface phase integral for common facet shapes. These will be used to derive closed-form expressions for the specular and backscatter RCS of PEC facets for these shapes. The surface phase integral is given by

$$I = \int_S e^{i\mathbf{K}\cdot\mathbf{r}} dS \tag{9.24}$$

where

$$\mathbf{K} = \mathbf{k}_i - \mathbf{k}_s \tag{9.25}$$

$$\mathbf{k}_i = k_i \left(\sin\theta_i \cos\phi_i \hat{x} + \sin\theta_i \sin\phi_i \hat{y} + \cos\theta_i \hat{z}\right) \tag{9.26}$$

$$\mathbf{k}_s = k_s \left(\sin\theta_s \cos\phi_s \hat{x} + \sin\theta_s \sin\phi_s \hat{y} + \cos\theta_s \hat{z}\right) \tag{9.27}$$

$$\mathbf{r} = x\hat{x} + y\hat{y} + z\hat{z} \tag{9.28}$$

The scattered wave vector, $\mathbf{k}_s$, can be a reflected or transmitted wave. If a facet is embedded in a large surface interface, the wavenumbers of the incident and scattered directions must correspond to correct medium above or below the interface.

The surface phase integral can be computed for facets that are tilted arbitrarily in a global frame. However, analytic expressions are more easily derived using facets in the XY plane, Figure 9.2, which is our approach in the following subsections. When a facet is tilted in the global frame, the incident and scattered angles need to be transformed to the local frame of the facet. This can be done by first having the forward rotation of the facet relative to the global frame, then applying the rotation in reverse to the incident and scattered wave vectors in the global frame, which will give the incident and scattered wave vectors in the facet frame.

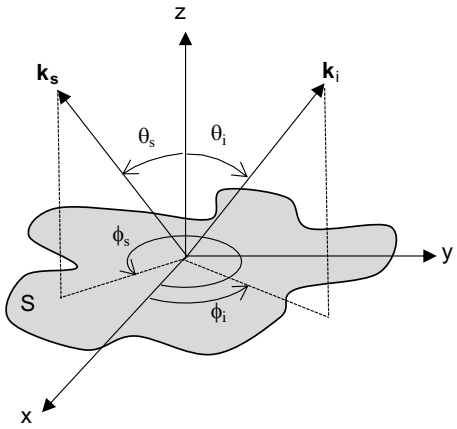


Figure 9.2: Geometry for the surface phase integral of a facet in the XY plane.

9.4.1 Specular Direction

Let the facet be in the XY plane with $\hat{n} = \hat{z}$. In the specular direction, $\theta_s = \pi - \theta_i$ and $\phi_s = \phi_i$, which means that the scattered direction is equivalent to the incident direction with the opposite z component. The wave vector difference is then

$$\mathbf{k}_i - \mathbf{k}_s = 2\cos\theta_i \hat{z} \tag{9.29}$$

The position vector in this case is $\mathbf{r} = x\hat{x} + y\hat{y}$. Therefore, $(\mathbf{k}_i - \mathbf{k}_s) \cdot \mathbf{r} = 0$, so the complex exponent evaluates to one, and

$$I = \int_S 1 \; dS = A \tag{9.30}$$

where A is the area of the facet. The surface phase integral in the specular direction is equal to the area of the facet. This is true for any shape.

9.4.2 Rectangle

Let a rectangular facet lie in the XY plane with side lengths L_x and L_y, $\hat{n} = \hat{z}$, and the facet is centered at the origin. The surface phase integral is written

$$I = \int_{-L_y/2}^{L_y/2} \int_{-L_x/2}^{L_x/2} e^{i\mathbf{K}\cdot\mathbf{r}} dx dy \tag{9.31}$$

$$\mathbf{K} = K_x\hat{x} + K_y\hat{y} + K_z\hat{z} \tag{9.32}$$

$$\mathbf{r} = x\hat{x} + y\hat{y} \tag{9.33}$$

where $\mathbf{K} = \mathbf{k}_i - \mathbf{k}_s$. The integral separates as

$$I = \int_{-L_x/2}^{L_x/2} e^{iK_x x} dx \int_{-L_y/2}^{L_y/2} e^{iK_y y} dy \tag{9.34}$$

$$\tag{9.35}$$

Using the fact that

$$\int_{-a/2}^{a/2} e^{ibz} dz = \frac{2}{b} \sin\left(\frac{ab}{2}\right) \tag{9.36}$$

and multiplying top and bottom by $L_x L_y$ to convert the sines to sinc functions, the surface phase integral over a rectangular facet is

$$I = L_x L_y \text{sinc}\left(\frac{L_x K_x}{2}\right) \text{sinc}\left(\frac{L_y K_y}{2}\right) \tag{9.37}$$

where $\text{sinc}(x) = \sin(x)/x$. This is equal to the area of the facet multiplied by a directionally dependent function that has maximum value of one.

9.4.3 Ellipse

Let the facet be an ellipse in the XY plane centered at the origin such that $\hat{n} = \hat{z}$ with semi-major axis a along the x axis, and semi-minor axis b along the y axis, with a contour described by the equation

$$\frac{x^2}{a^2} + \frac{y^2}{b^2} = 1 \tag{9.38}$$

Using the change of variables

$$x = a\rho\cos\phi \tag{9.39}$$

$$y = b\rho\sin\phi \tag{9.40}$$

$$dS = ab\rho d\rho d\phi \tag{9.41}$$

$$\mathbf{r} = a\rho\cos\phi\hat{x} + b\rho\sin\phi\hat{y} \tag{9.42}$$

we transform (9.24) to an integration in polar coordinates over the surface of the ellipse as

$$I = ab \int_0^{2\pi} \int_0^1 e^{i(K_x a\rho\cos\phi + K_y b\rho\sin\phi)} \rho d\rho d\phi \tag{9.43}$$

Applying the following identity

$$\int_0^{2\pi} e^{u\cos t + v\sin t} dt = 2\pi I_0\left(\sqrt{u^2+v^2}\right) \tag{9.44}$$

we get

$$I = 2\pi ab \int_0^1 I_0\left(i\rho\sqrt{(aK_x)^2+(bK_y)^2}\right)\rho d\rho \tag{9.45}$$

Finally, using

$$\int_0^1 I_0\left(icx\right) x dx = \frac{J_1(c)}{c} \tag{9.46}$$

the surface phase integral over an ellipse is

$$I = 2\pi ab \frac{J_1(K'_\rho)}{K'_\rho} \tag{9.47}$$

$$= 2\pi ab \text{jinc}(K'_\rho) \tag{9.48}$$

$$K'_\rho = \sqrt{(aK_x)^2+(bK_y)^2} \tag{9.49}$$

Note, jinc(0) = 1/2, so the final result is equal to the area of the ellipse multiplied by a directionally dependent term that has maximum value of one. K'_ρ is a modified perpendicular component of the wave vector difference.

9.4.4 Circle

The surface phase integral for the circular disk in the XY plane with radius a can be derived directly from the elliptical disk, (9.48), by taking $b = a$. This gives

$$I = 2\pi a^2 \text{jinc}\left(aK_\rho\right) \tag{9.50}$$

$$K_\rho = \sqrt{K_x^2 + K_y^2} \tag{9.51}$$

A solution for the circular disk can also be found in [49], which uses a sequence of trigonometric transformations to separate the magnitude and phase of the perpendicular wave vector difference.

9.4.5 Triangle

Let the facet be a triangle in the XY plane with $\hat{n} = \hat{z}$. The vertices of the triangle are at positions $[\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3]$. The vector to the triangle centroid is given by

$$\mathbf{r}_n = \frac{1}{3}\left[\mathbf{p}_1 + \mathbf{p}_2 + \mathbf{p}_3\right] \tag{9.52}$$

The phase integral is computed by mapping the domain of the arbitrary triangular facet to a triangle that covers half of the unit square though the following transform, [50],

$$x(u,v) = x_1 + u(x_2 - x_1) + v(x_3 - x_1) \tag{9.53}$$

$$y(u,v) = y_1 + u(y_2 - y_1) + v(y_3 - y_1) \tag{9.54}$$

where $u = [0, 1]$ and $v = [0, 1 - u]$. $\mathbf{p}_1$ is mapped to the origin, $\mathbf{p}_2$ is mapped to $(1, 0)$ and $\mathbf{p}_3$ is mapped to $(0, 1)$. The position vector is parameterized in terms of u and v as

$$\mathbf{r} = \mathbf{p}_1 + u(\mathbf{p}_2 - \mathbf{p}_1) + v(\mathbf{p}_3 - \mathbf{p}_1) \tag{9.55}$$

The change of variables in the surface integral is

$$I = \int_S f(x(u, v), y(u, v))|\mathbf{J}(u, v)|dudv \tag{9.56}$$

The determinant of the Jacobian is

$$\begin{aligned} |\mathbf{J}(u, v)| &= \det \begin{bmatrix} \partial x/\partial u & \partial x/\partial v \\ \partial y/\partial u & \partial y/\partial v \end{bmatrix} && (9.57) \\ &= (x_2 - x_1)(y_3 - y_1) - (x_3 - x_1)(y_2 - y_1) && (9.58) \\ &= 2A && (9.59) \end{aligned}$$

which is equal to the twice the area of the triangle. With these, the integral becomes

$$I = 2A \int_0^1 \int_0^{1-u} e^{i(a+bu+cv)} dvdu \tag{9.60}$$

where

$$\begin{aligned} a &= \mathbf{K} \cdot \mathbf{p}_1 && (9.61) \\ b &= \mathbf{K} \cdot (\mathbf{p}_2 - \mathbf{p}_1) && (9.62) \\ c &= \mathbf{K} \cdot (\mathbf{p}_3 - \mathbf{p}_1) && (9.63) \end{aligned}$$

Integrating this yields

$$\begin{aligned} I(a, b, c) &= Ae^{ia} g(b, c) && (9.64) \\ g(b, c) &= 2\frac{c(1 - e^{ib}) - b(1 - e^{ic})}{bc(b - c)} && (9.65) \end{aligned}$$

Divide by zero occurs for $b = 0$, $c = 0$, or $b = c$. The first two cases happen when the wave vector difference is perpendicular to an edge of the triangle. The third happens when the wave vector difference bisects the angle formed by two of the edges with $\mathbf{p}_1$ at the vertex. The limits, though, exist and are given by

$$\begin{aligned} \lim_{b \to 0} g(b, c) &= 2\frac{1 + ic - e^{ic}}{c^2} && (9.66) \\ \lim_{c \to 0} g(b, c) &= 2\frac{1 + ib - e^{ib}}{b^2} && (9.67) \\ \lim_{c \to b} g(b, c) &= 2\frac{-1 + (1 - ib)e^{ib}}{b^2} && (9.68) \\ \lim_{[b,c] \to [0,0]} g(b, c) &= 1 && (9.69) \end{aligned}$$

9.5 S-matrix of a Facet

Here we develop the idea of an S-matrix for a Kirchhoff facet. The flat facet is treated as an isolated scatterer. When the facet is part of a larger surface discretization, the facet is assumed to be non-interacting with neighboring facets and which has homogeneous and semi-infinite mediums above and below it. We derive the general cases for the scattering of a facet and also provide expressions for their radar cross sections.

Define the second orthonormal system $(\hat{p}_s, \hat{q}_s, \hat{k}_s)$ for the scattered field, which is relative to the surface of the facet and applies to both the reflected and transmitted fields,

$$\hat{q}_s = \frac{\hat{k}_s \times \hat{n}}{|\hat{k}_s \times \hat{n}|} \tag{9.70}$$

$$\hat{p}_s = \hat{q}_s \times \hat{k}_s \tag{9.71}$$

The unit vectors that make up the identity dyad are defined in terms of the scattered field system, $\bar{\mathbf{I}} = \hat{q}_s\hat{q}_s + \hat{p}_s\hat{p}_s + \hat{k}_s\hat{k}_s$, so that the reflected and transmitted fields for a single facet are

$$\mathbf{E}_r(\mathbf{r}) \approx \frac{ik_1 e^{ik_1 r}}{4\pi r} E_o \left(\hat{q}_r\hat{q}_r + \hat{p}_r\hat{p}_r\right) \cdot \mathbf{F}(\hat{k}_i, \hat{k}_r) I(\mathbf{k}_i, \mathbf{k}_r) \tag{9.72}$$

$$\mathbf{E}_t(\mathbf{r}) \approx -\frac{ik_2 e^{ik_2 r}}{4\pi r} E_o \left(\hat{q}_t\hat{q}_t + \hat{p}_t\hat{p}_t\right) \cdot \mathbf{N}(\hat{k}_i, \hat{k}_t) I(\mathbf{k}_i, \mathbf{k}_t) \tag{9.73}$$

where I is the surface phase integral (9.23). Define two scattering matrices, S_{11} and S_{21}, which map plane waves incident from medium 1 to far-field plane waves that are reflected or transmitted waves in mediums 1 and 2, respectively. S_{11} is valid above the interface, while S_{21} is valid below the interface. Writing the incident, reflected and transmitted fields in the $\hat{p}$, $\hat{q}$ polarization basis

$$\mathbf{E}_r = \frac{e^{ik_1 r}}{r} \mathbf{S}_{11} \cdot \mathbf{E}_i \tag{9.74}$$

$$\mathbf{E}_t = \frac{e^{ik_2 r}}{r} \mathbf{S}_{21} \cdot \mathbf{E}_i \tag{9.75}$$

$$\mathbf{E}_i = E_o \begin{bmatrix} (\hat{e}_i \cdot \hat{p}_i) \\ (\hat{e}_i \cdot \hat{q}_i) \end{bmatrix} \tag{9.76}$$

where the elements of the reflected and transmitted S-matrices are

$$\mathbf{S}_{11} = \frac{ik_1}{4\pi} I(\mathbf{k}_i, \mathbf{k}_r, \hat{n}) \begin{bmatrix} \hat{p}_r \cdot \mathbf{F}_p & \hat{p}_r \cdot \mathbf{F}_q \\ \hat{q}_r \cdot \mathbf{F}_p & \hat{q}_r \cdot \mathbf{F}_q \end{bmatrix} \tag{9.77}$$

$$\mathbf{S}_{21} = -\frac{ik_2}{4\pi} I(\mathbf{k}_i, \mathbf{k}_t, \hat{n}) \begin{bmatrix} \hat{p}_t \cdot \mathbf{N}_p & \hat{p}_t \cdot \mathbf{N}_q \\ \hat{q}_t \cdot \mathbf{N}_p & \hat{q}_t \cdot \mathbf{N}_q \end{bmatrix} \tag{9.78}$$

and

$$\mathbf{F}_p = (\hat{n} \times \hat{q}_i)(1 + R^{\mathrm{TM}}) + (\hat{n} \cdot \hat{k}_i)(\hat{k}_r \times \hat{q}_i)(1 - R^{\mathrm{TM}}) \tag{9.79}$$

$$\mathbf{F}_q = -(\hat{n} \cdot \hat{k}_i)\hat{q}_i(1 - R^{\mathrm{TE}}) + (\hat{k}_r \times (\hat{n} \times \hat{q}_i))(1 + R^{\mathrm{TE}}) \tag{9.80}$$

$$\mathbf{N}_p = \frac{\eta_2}{\eta_1}(\hat{n} \times \hat{q}_i)(1 + R^{\mathrm{TM}}) + (\hat{n} \cdot \hat{k}_i)(\hat{k}_t \times \hat{q}_i)(1 - R^{\mathrm{TM}}) \tag{9.81}$$

$$\mathbf{N}_q = -\frac{\eta_2}{\eta_1}(\hat{n} \cdot \hat{k}_i)\hat{q}_i(1 - R^{\mathrm{TE}}) + (\hat{k}_t \times (\hat{n} \times \hat{q}_i))(1 + R^{\mathrm{TE}}) \tag{9.82}$$

Table 9.1 gives the dot products between $\hat{p}$ and $\hat{q}$ and the different vector quantities listed at the top of the columns.

Table 9.1: Table of vector products

	$\hat{n} \times \hat{q}_i$	$\hat{k} \times \hat{q}_i$	$\hat{q}_i$	$\hat{k} \times (\hat{n} \times \hat{q}_i)$
$\hat{p}\cdot$	$\hat{n} \cdot (\hat{q}_i \times \hat{p})$	$-\hat{q}_i \cdot \hat{q}$	$\hat{p} \cdot \hat{q}_i$	$(\hat{k} \cdot \hat{q}_i)(\hat{p} \cdot \hat{n}) - (\hat{k} \cdot \hat{n})(\hat{q}_i \cdot \hat{p})$
$\hat{q}\cdot$	$\hat{n} \cdot (\hat{q}_i \times \hat{q})$	$\hat{p} \cdot \hat{q}_i$	$\hat{q} \cdot \hat{q}_i$	$-(\hat{k} \cdot \hat{n})(\hat{q} \cdot \hat{q}_i)$

Using these, we can write

$$\hat{p}_r \cdot \mathbf{F}_p = \hat{n} \cdot (\hat{q}_i \times \hat{p}_r)(1 + R^{\mathrm{TM}}) + (\hat{n} \cdot \hat{k}_i)(-\hat{q}_i \cdot \hat{q}_r)(1 - R^{\mathrm{TM}}) \tag{9.83}$$

$$\hat{q}_r \cdot \mathbf{F}_p = \hat{n} \cdot (\hat{q}_i \times \hat{q}_r)(1 + R^{\mathrm{TM}}) + (\hat{n} \cdot \hat{k}_i)(\hat{q}_i \cdot \hat{p}_r)(1 - R^{\mathrm{TM}}) \tag{9.84}$$

$$\hat{p}_r \cdot \mathbf{F}_q = -(\hat{n} \cdot \hat{k}_i)(\hat{p}_r \cdot \hat{q}_i)(1 - R^{\mathrm{TE}}) + ((\hat{k}_r \cdot \hat{q}_i)(\hat{p}_r \cdot \hat{n}) - (\hat{k}_r \cdot \hat{n})(\hat{q}_i \cdot \hat{p}_r))(1 + R^{\mathrm{TE}}) \tag{9.85}$$

$$\hat{q}_r \cdot \mathbf{F}_q = -(\hat{n} \cdot \hat{k}_i)(\hat{q}_r \cdot \hat{q}_i)(1 - R^{\mathrm{TE}}) - (\hat{k}_r \cdot \hat{n})(\hat{q}_r \cdot \hat{q}_i)(1 + R^{\mathrm{TE}}) \tag{9.86}$$

The same applies to $\mathbf{N}$ except with $r \to t$ and the inclusion of material constants.

9.6 Radar Cross Section of a Facet

Recall that the polarized bistatic radar cross section is defined as

$$\sigma_{pq}(\hat{k}_s, \hat{k}_i) = \lim_{kr \to \infty} 4\pi r^2 \frac{|\hat{p} \cdot \mathbf{E}_s|^2}{|\hat{q} \cdot \mathbf{E}_i|^2} \tag{9.87}$$

$$= 4\pi |S_{pq}|^2 \tag{9.88}$$

The total radar cross section for either scattered polarization is

$$\sigma_p = \sigma_{pp}(\hat{e}_i \cdot \hat{p}_i)^2 + \sigma_{pq}(\hat{e}_i \cdot \hat{q}_i)^2 \tag{9.89}$$

$$\sigma_q = \sigma_{qp}(\hat{e}_i \cdot \hat{p}_i)^2 + \sigma_{qq}(\hat{e}_i \cdot \hat{q}_i)^2 \tag{9.90}$$

In the frame of the facet, $\hat{q}$ and $\hat{p}$ are equivalent to the traditional $\hat{h}$ and $\hat{v}$ polarizations. In other words, if $\hat{n} = \hat{z}$, then $\hat{q} = \hat{h}$ and $\hat{p} = \hat{v}$ in a global frame.

9.6.1 Specular RCS of a Facet

The special case of the facet RCS in the specular direction can be derived using (9.77). Assume that the facet lies in the XY plane with $\hat{n} = \hat{z}$. In the specular direction, we have $\theta_r = \pi - \theta_i$ and $\phi_r = \phi_i$, where the reflected wave vector is just the incidence wave vector reflected from the XY plane. This means that

$$\hat{q}_r = \hat{q}_i \tag{9.91}$$

$$\hat{q}_i \cdot \hat{q}_r = 1 \tag{9.92}$$

$$\hat{p}_r \cdot \hat{q}_i = 0 \tag{9.93}$$

$$\hat{q}_i \times \hat{p}_r = -\hat{k}_r \tag{9.94}$$

$$\hat{q}_i \times \hat{q}_r = 0 \tag{9.95}$$

$$\hat{q}_i \times \hat{p}_r = 0 \tag{9.96}$$

Using these in (9.83)-(9.86) gives:

$$\hat{p}_r \cdot \mathbf{F}_p = \hat{n} \cdot (-\hat{k}_r)(1 + R^{\mathrm{TM}}) - (\hat{n} \cdot \hat{k}_i)(1 - R^{\mathrm{TM}}) \tag{9.97}$$

$$\hat{q}_r \cdot \mathbf{F}_p = 0 \tag{9.98}$$

$$\hat{p}_r \cdot \mathbf{F}_q = 0 \tag{9.99}$$

$$\hat{q}_r \cdot \mathbf{F}_q = -(\hat{n} \cdot \hat{k}_i)(1 - R^{\mathrm{TE}}) - (\hat{k}_r \cdot \hat{n})(1 + R^{\mathrm{TE}}) \tag{9.100}$$

Using θ_i as the local incidence angle, we have for the specular direction $\cos\theta_i = -\hat{n}\cdot\hat{k}_i$ and $\cos\theta_i = \hat{n}\cdot\hat{k}_r$. Substituting these and simplifying, we can write the polarized RCSs as

$$\sigma_{pp} = \sigma_{pec}|R^{\mathrm{TM}}|^2 \tag{9.101}$$
$$\sigma_{qq} = \sigma_{pec}|R^{\mathrm{TE}}|^2 \tag{9.102}$$
$$\sigma_{pq} = \sigma_{qp} = 0 \tag{9.103}$$

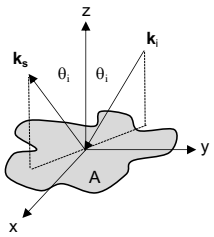
where

$$\sigma_{pec} = \frac{k^2}{\pi}\cos^2\theta_i |I(\mathbf{k}_i, \mathbf{k}_r)|^2 \tag{9.104}$$

and σ_{pec} is the RCS for a PEC surface, evaluated in the specular direction. This quantity assumes perfect reflection, and θ_i is the local incidence angle measured from the facet normal. These show that 1) there are no cross polarization terms for the specular direction of a flat facet under the Kirchhoff approximation, and 2) the specular RCS of a facet at a dielectric interface is the same as a PEC facet only scaled by the reflectivity of the surface evaluated at the incidence angle. Derived above, the value of $|I|^2$ in the specular direction is equal to the area of the facet squared. This is true regardless of the shape. Therefore, in the specular direction, the RCS of a PEC facet is just

$$\sigma_{pec} = \frac{k^2}{\pi}\cos^2\theta_i A^2 \tag{9.105}$$

Table 9.2: Specular RCS of a PEC Facet

Facet	Geometry	σ_{pec}	Notes
Any shape		$\frac{k^2}{\pi}\cos^2\theta_i A^2$	A is the facet area

9.6.2 Backscatter RCS of a Facet

To obtain the backscatter RCS of a facet, we use (9.77) with $\hat{k}_r = -\hat{k}_i$, $\hat{q}_r = -\hat{q}_i$, $\hat{p}_r = \hat{p}_i$. Substituting these into (9.83)-(9.86) it can be shown that

$$\hat{p}_r\cdot\mathbf{F}_p = -2(\hat{n}\cdot\hat{k}_i)R^{\mathrm{TM}} \tag{9.106}$$
$$\hat{q}_r\cdot\mathbf{F}_p = 0 \tag{9.107}$$
$$\hat{p}_r\cdot\mathbf{F}_q = 0 \tag{9.108}$$
$$\hat{q}_r\cdot\mathbf{F}_q = -2(\hat{n}\cdot\hat{k}_i)R^{\mathrm{TE}} \tag{9.109}$$

Then using (9.77) and (9.88), we get

$$\sigma_{pp} = \sigma_{pec}|R^{\mathrm{TM}}|^2 \tag{9.110}$$
$$\sigma_{qq} = \sigma_{pec}|R^{\mathrm{TE}}|^2 \tag{9.111}$$
$$\sigma_{pq} = \sigma_{qp} = 0 \tag{9.112}$$

where

$$\sigma_{pec} = \frac{k^2}{\pi}\cos^2\theta_i |I(\mathbf{k}_i, -\mathbf{k}_i)|^2 \tag{9.113}$$

where σ_{pec} is the backscatter RCS for a PEC surface. As with the specular direction, this assumes perfect reflection, and θ_i is the local incidence angle measured from the facet normal. There are no cross polarization terms for the backscatter of a flat facet under the Kirchhoff approximation, and the backscatter RCS of a facet at a dielectric interface is equal to the backscatter RCS of a PEC facet scaled by the reflectivity. The surface phase integral has to be evaluated over the facet shape. We use the results from Section 9.4 to find σ_{pec} in the backscatter direction for simple shapes.

Rectangle Assuming that the facet is in the XY plane with $\hat{n} = \hat{z}$. In the backscatter direction $k_i = k_s = k$, $\theta_s = \pi - \theta_i$ and $\phi_s = \phi_i + \pi$, and it can be shown that the components of the wave vector difference is equal to

$$K_x = 2k\sin\theta_i\cos\phi_i \tag{9.114}$$
$$K_y = 2k\sin\theta_i\sin\phi_i \tag{9.115}$$

The surface phase integral in over a rectangle, (9.37), in the backscatter direction is

$$I = L_x L_y \text{sinc}\left(L_x k\sin\theta_i\cos\phi_i\right)\text{sinc}\left(L_y k\sin\theta_i\sin\phi_i\right) \tag{9.116}$$

Using this in (9.113), the backscatter RCS of a rectangular PEC facet is

$$\sigma_{pec} = \frac{k^2}{\pi}\cos^2\theta_i L_x^2 L_y^2 \text{sinc}^2\left(L_x k\sin\theta_i\cos\phi_i\right)\text{sinc}^2\left(L_y k\sin\theta_i\sin\phi_i\right) \tag{9.117}$$

In (9.113), $\cos\theta_i$ was defined for the local incident angle using (9.14). However, the angles in the argument of the sinc technically belong to the incident wave vector. Because $\text{sinc}(x)$ is even, and because $\theta_i \to \pi - \theta_i$ and $\phi_i \to \phi_i + \pi$, the result is the same. Therefore, the angles can be treated as belonging to either the incident wave vector or a vector that points to the sensor for the case of backscatter. Note, the RCS is proportional to the area of the facet squared.

Ellipse Assuming that the facet is in the XY plane, and with (9.114) and (9.115), the surface phase integral over an ellipse, (9.48) reduces to

$$I = 2\pi ab\,\text{jinc}\left(2k\sin\theta_i\sqrt{a^2\cos^2\phi_i + b^2\sin^2\phi_i}\right) \tag{9.118}$$

Using this in the expression for radar cross section for a PEC surface, (9.113), the backscatter RCS of a PEC elliptical disk is

$$\sigma_{pec} = 4\pi k^2\cos^2\theta_i a^2 b^2 \text{jinc}^2\left(2k\sin\theta_i\sqrt{a^2\cos^2\phi_i + b^2\sin^2\phi_i}\right) \tag{9.119}$$

Because this is an even function of θ_i and ϕ_i, the angles can be defined either from the incident wave vector or from a vector that points to the sensor and measured from the facet normal.

Circle Using (9.118) with $b = a$, the surface phase integral for a circular disk is

$$I = 2\pi a^2\,\text{jinc}\left(2k\sin\theta_i a\right) \tag{9.120}$$

Likewise, from (9.119), the backscatter RCS of the circular PEC disk is

$$\sigma_{pec} = 4\pi k^2 a^4\cos^2\theta_i \text{jinc}^2\left(2ka\sin\theta_i\right) \tag{9.121}$$

Again, θ_i can be either the forward incident angle, or the local incident angle from facet normal.

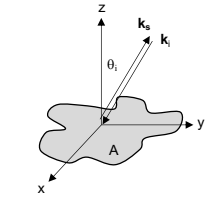
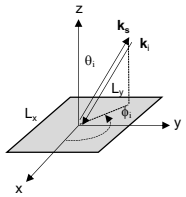
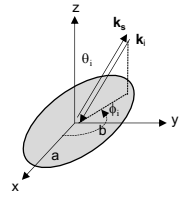
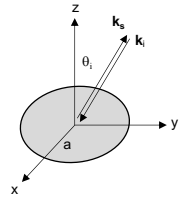
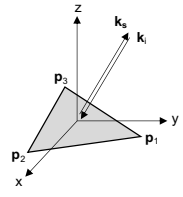
Triangle For the backscatter direction, $\mathbf{K} = 2\mathbf{k}_i$. When $\mathbf{K}$ is real, the surface phase integral of a triangular facet, (9.64), has the following identity $I(-a,-b,-c) = I^*(a,b,c)$. Therefore, reversing the incident direction to use angles that point at the sensor will not change $|I|^2$. The backscatter RCS of a PEC triangle is then

$$\sigma_{pec} = \frac{k^2}{\pi} A^2 \cos^2\theta_i |g(b,c)|^2 \tag{9.122}$$

No simple reduction has been found for $|g(b,c)|^2$.

Summary Table 9.3 has a summary of σ_{pec} for these shapes. In general, σ_{pec} can be decomposed as a product of four terms: 1) a factor of k^2/π, 2) a factor of $\cos^2\theta_i$ for the area projection, 3) a factor of the facet area, and 4) a directionally dependent weighting function that has maximum value of one and comes from the Fourier transform over the domain of the facet via the surface phase integral.

Table 9.3: Backscatter RCS of PEC Facets

Facet	Geometry	σ_{pec}	Notes
Any shape		$\frac{k^2}{\pi}\cos^2\theta_i \lvert I(\mathbf{k}_i,-\mathbf{k}_i)\rvert^2$	
Rectangle		$\frac{k^2}{\pi}\cos^2\theta_i L_x^2 L_y^2 \cdot$ $\operatorname{sinc}^2(L_x k \sin\theta_i \cos\phi_i)\operatorname{sinc}^2(L_y k \sin\theta_i \sin\phi_i)$	$\operatorname{sinc}(x) = \frac{\sin(x)}{x}$
Ellipse		$4\pi k^2 \cos^2\theta_i a^2 b^2 \cdot$ $\operatorname{jinc}^2\left(2k\sin\theta_i\sqrt{a^2\cos^2\phi_i + b^2\sin^2\phi_i}\right)$	$\operatorname{jinc}(x) = \frac{J_1(x)}{x}$
Circle		$4\pi k^2 a^4 \cos^2\theta_i \operatorname{jinc}^2(2ka\sin\theta_i)$	
Triangle		$\frac{k^2}{\pi} A^2 \cos^2\theta_i \lvert g(b,c)\rvert^2$	$b = \mathbf{K}\cdot(\mathbf{p}_2 - \mathbf{p}_1)$ $c = \mathbf{K}\cdot(\mathbf{p}_3 - \mathbf{p}_1)$

Chapter 10

Random Object Generation

Generating random signals, surfaces, and volumes is important for the study of stochastic scattering processes. Stationary Gaussian random signals are fully characterized by the correlation function or, equivalently, the power spectral density (PSD), which take two parameters: root-mean-square (RMS) height and correlation length. The correlation function can be anisotropic for two and three dimensions. The PSD is the average power envelope of a randomized frequency spectrum. To generate a random signal or surface from a PSD, each frequency of the sampled PSD is seeded with a complex standard Gaussian, and then an inverse Fourier transform is taken. Derivations of isotropic Gaussian and exponential PSDs in one, two, and three dimensions can be found in [51], which can be extended to the anisotropic cases.

Fractal surfaces are different creatures. They are Gaussian processes that exhibit self-similarity at different length scales. This means that the RMS height changes depending on the distance over which one measures it. The two parameters that characterize fractal surfaces are the RMS height (or RMS variation) at a given length scale and a Hurst exponent. Fractal surfaces are considered better representations of naturally occurring surfaces than those given by the exponential or Gaussian correlation functions.

In general, large domains and high sampling rates are required to generate scenes with correct statistics. The scene size should contain many correlation lengths and there should be a sufficient number of points to sample the highest spectral content. When these conditions are not met, the RMS of numerically generated signals is usually lower than the desired value, [52]. A quick fix is to simply renormalize the surface RMS to the desired value. When the correlation length is less than the sample rate, or when the correlation length is equal to zero, surface points can simply be drawn as independent Gaussian random variables with a desired RMS.

We give routines for generating 1D, 2D, and 3D Gaussian random signals, surfaces, and volumes with Gaussian or exponential correlation functions, 1D fractal surfaces, as well as 3D bicontinuous media. We give a routine for generating ocean waves that have nonlinear dispersion relations. In addition, we provide a routine for generating spherical particles where the surface radius has log-normal statistics. Finally, a routine for creating 1D profiles of ionosphere irregularity is included.

10.1 1D Random Signals

A 1D random signal could be a voltage signal, or a corrugated rough surface (like a corn field) that have variation in only one dimension. The process of generating a surface from a power spectral density is detailed in [52, 53]. The autocorrelation of a zero mean stationary random process f is given by

$$< f(x_1)f(x_2) >= \sigma^2 C(x_2 - x_1) \tag{10.1}$$

where σ is the root-mean-squared (RMS) variation and $C(\Delta x)$ is the correlation function. Here the correlation function is only a function of the difference of distances between points, or lag, $\Delta x = x_2 - x_1$. Gaussian

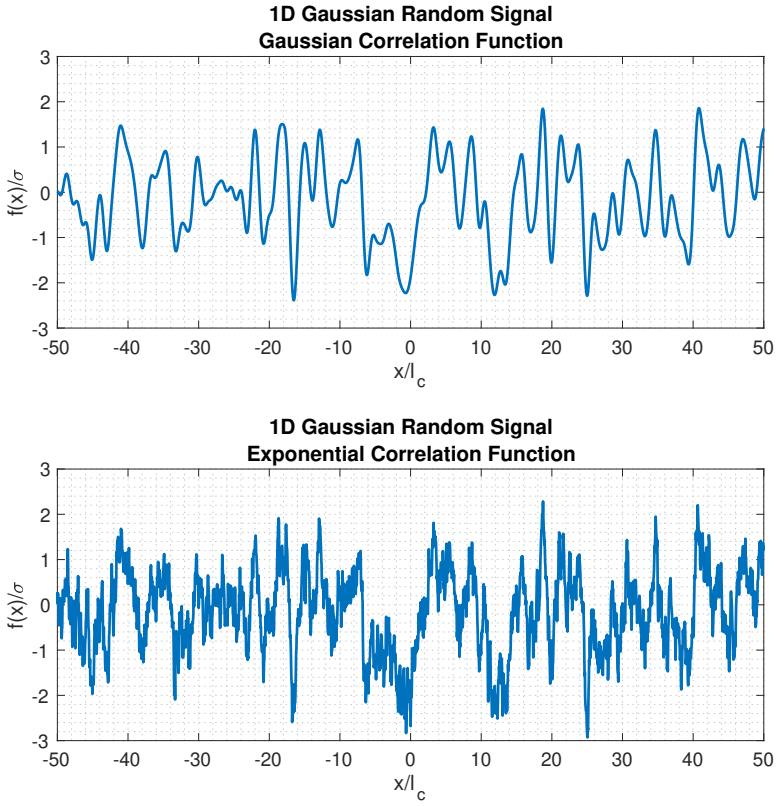


Figure 10.1: 1D Gaussian random signals with Gaussian and exponential correlation functions generated from the same random seed. The vertical and horizontal axes are normalized by the RMS and correlation length, respectively. Note, signals with an exponential correlation function always have a discontinuous derivative no matter how finely they are sampled.

and exponential correlation functions are given by

$$C(\Delta x) = \exp\left(-\frac{\Delta x^2}{l_c^2}\right) \tag{10.2}$$

$$C(\Delta x) = \exp\left(-\frac{|\Delta x|}{l_c}\right) \tag{10.3}$$

with correlation length l_c. The corresponding power spectral densities are, respectively,

$$W(k_x) = \frac{\sigma^2 l_c}{2\sqrt{\pi}} \exp\left(-\frac{k_x^2 l_c^2}{4}\right) \tag{10.4}$$

$$W(k_x) = \frac{\sigma^2 l_c}{\pi\left(1 + k_x^2 l_c^2\right)} \tag{10.5}$$

The spectrum of a 1D rough surface with physical length L_x is given by

$$F(k_x) = \gamma(k_x)\sqrt{\delta k_x W(k_x)} \tag{10.6}$$

where $\delta k_x = 2\pi/L_x$ and $\gamma(k_x)$ is an independent draw from the complex standard normal distribution for each spatial frequency k_x. It is γ that imparts randomness to each realization of the PSD. The complex standard normal is defined:

$$\gamma = \frac{1}{\sqrt{2}}\left(\mathcal{N}(0,1) + i\mathcal{N}(0,1)\right) \tag{10.7}$$

where $\mathcal{N}(0,1)$ is the standard normal distribution with zero mean and unit variance. γ is normalized so that its variance is one. Finally, the surface is realized with an inverse Fourier transform:

$$f(x) = \mathcal{F}_{k_x}^{-1}\left[F(k_x)\right] \tag{10.8}$$

Another important quantity is the variance of the difference of surface heights as a function of lag. This is called the Allan variance. The square root of this quantity is the Allan deviation or RMS deviation. The deviation is zero at zero lag and grows following the correlation function until points separated by more than the correlation length become uncorrelated and the variance is equal to two times the variance of the underlying process. The variance at lag Δx is

$$v^2(\Delta x) = 2\sigma^2\left(1 - C(\Delta x)\right) \tag{10.9}$$

Then the RMS deviation at lag Δx is

$$v(\Delta x) = \sqrt{2}\sigma\sqrt{(1 - C(\Delta x))} \tag{10.10}$$

The routine `rough1` generates a real-valued zero-mean 1D Gaussian random signal. The inputs are the number of points N_x, length of the signal L_x, RMS height σ, correlation length l_c. Use string switch `norm` or `exp` for Gaussian or exponential correlation functions. When the correlation length is less than the sample size, and therefore the signal is not properly sampled, the routine returns independent draws from a Gaussian distribution at each point with the given RMS. When generating a real valued signal from its spectrum, we usually make the spectrum conjugate symmetric before taking the IFFT. Instead, it is easier to seed the PSD with samples from an unnormalized non-conjugate complex standard normal distribution and then take the real part of the IFFT. Matlab's `ifft` divides by the number of samples in each dimension, so to keep the transform independent of domain size, this factor has to be put back. The 1D PSDs are coded inline, but can be plotted separately by the routines `psdnorm` and `psdexp`.

```
function h = rough1(Nx,Lx,rmsh,lc,type)
% Generate 1D random signal
%
% N:        N number of points
% L:        Physical length
% rmsh:     root mean square height
% lc:       correlation length
% type:     'norm' = Gaussian power spectral density
%           'exp' = Exponential power spectral density
%
% h:        Nx1 array of heights
%
% Dependencies: fftfreq

dx = Lx/(Nx-1);
if lc < dx
    h = rmsh*randn(Nx,1);
else
    kx = 2*pi*fftfreq(1/dx,Nx);
    if strcmp(type,'norm'),
         W = rmsh^2*lc*exp(-(kx*lc*0.5).^2)/(2*sqrt(pi));
    elseif strcmp(type,'exp'),
         W = rmsh^2*lc./(pi*(1+(kx*lc).^2));
    else
        error('bad PSD type')
    end
    gam = randn(Nx,1) + 1i*randn(Nx,1);
    gam(1) = 0;
    H = gam.*sqrt(W);
    dkx = 2*pi/Lx;
    h = (Nx*sqrt(dkx))*real(ifft(H));
end
```

10.2 2D Random Surfaces

A 2D random rough surface is like a beach or patch of dirt, which is a height map of two variables. 2D rough surfaces are created like 1D random signals, except that the PSDs acquire an extra dimension and scaling factors. In addition, anisotropic correlations are now possible. Gaussian and exponential correlation functions are given by

$$C(\Delta x, \Delta y) = \exp\left(-\frac{\Delta x^2}{l_{cx}^2} - \frac{\Delta y^2}{l_{cy}^2}\right) \tag{10.11}$$

$$C(\Delta x, \Delta y) = \exp\left(-\sqrt{\frac{\Delta x^2}{l_{cx}^2} + \frac{\Delta y^2}{l_{cy}^2}}\right) \tag{10.12}$$

The corresponding power spectral densities are

$$W(k_x, k_y) = \frac{\sigma^2 l_{cx} l_{cy}}{4\pi} \exp\left(-\frac{k_x^2 l_{cx}^2}{4} - \frac{k_y^2 l_{cy}^2}{4}\right) \tag{10.13}$$

$$W(k_x, k_y) = \frac{\sigma^2 l_{cx} l_{cy}}{2\pi\left(1 + k_x^2 l_{cx}^2 + k_y^2 l_{cy}^2\right)^{3/2}} \tag{10.14}$$

The 2D frequency spectrum is given by

$$F(\mathbf{k}) = \gamma(\mathbf{k})\sqrt{\delta k_x \delta k_y W(\mathbf{k})} \tag{10.15}$$

$$\delta k_x = \frac{2\pi}{L_x} \quad \delta k_y = \frac{2\pi}{L_y} \quad k = \sqrt{k_x^2 + k_y^2} \tag{10.16}$$

Here, $\gamma(\mathbf{k})$ is the complex standard normal distribution evaluated independently at each $\mathbf{k}$.

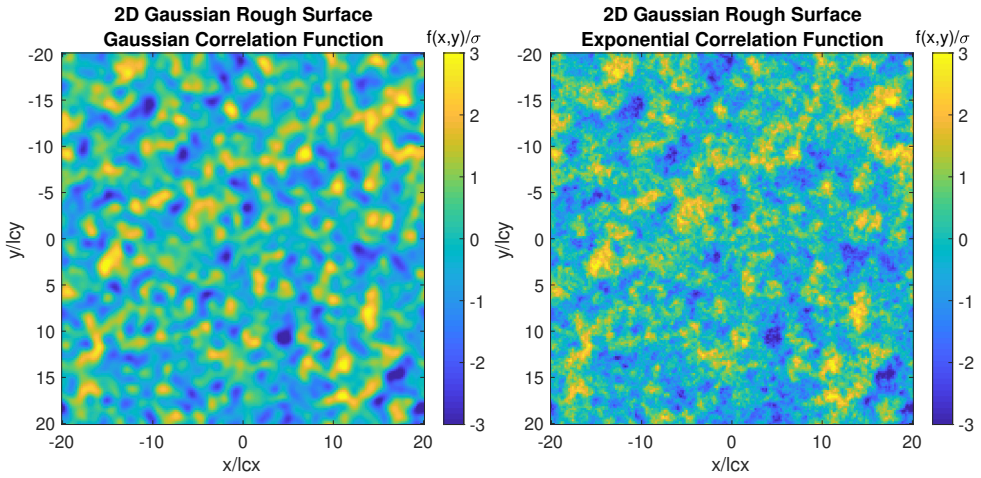


Figure 10.2: 2D Gaussian and exponentially correlated rough surfaces generated with the same random seed. The surface height is normalized by the RMS, and the axes are normalized by the correlation length.

The routine `rough2` generates a real-valued zero-mean 2D Gaussian random surface. The inputs are the number of points N_x, N_y in each dimension, side lengths of the domain L_x, L_y, RMS height σ, and correlation lengths in each dimension l_{cx}, l_{cy}. It returns the surface in `meshgrid` format. Use string switch `norm` or `exp` for Gaussian or exponential correlation functions. As in the 1D case, we can avoid having to make the spectrum conjugate symmetric by drawing from the unnormalized non-conjugate complex standard normal and then simply taking the real part of the 2D IFFT. If the correlation lengths in both dimensions are less

than their respective sample sizes, the surface heights are independent draws from a Gaussian distribution. If the correlation length of one dimension is less than its sample size, then we treat the other dimension as a collection of independent 1D rough surfaces. These provisions help maintain the correct RMS height when the PSD is poorly sampled.

```
function [h] = rough2(Nx,Ny,Lx,Ly,rmsh,lcx,lcy,type)
% Generate 2D random rough surface
%
%  Nx,Ny:       number of points in x and y
%  Lx,Ly:       physical extent of x and y
%  rmsh:        root mean square height
%  lcx,lcy:     correlation lengths in x and y
%  type:        'norm' = Gaussian power spectral density
%               'exp' = Exponential power spectral density
%
%  h:           [Ny,Nx] meshgrid array of heights
%
%  Dependencies: fftfreq

if strcmp(type,'norm')
    typesw = 1;
elseif strcmp(type,'exp')
    typesw = 0;
else
   error('bad PSD type')
end
dx = Lx/(Nx-1);
dy = Ly/(Ny-1);
dd = [dx dy];
Ns = [Ny Nx];
Ls = [Ly Lx];
Lcs = [lcy lcx];
undersamp = [(lcy<dy) (lcx<dx)];
% all correlation lengths are well sampled
if sum(undersamp) == 0
    kx = 2*pi*fftfreq(1/dx,Nx);
    ky = 2*pi*fftfreq(1/dy,Ny);
    [Kx Ky] = meshgrid(kx,ky);
    if typesw
        W = (rmsh^2*lcx*lcy/4/pi)*exp(-lcx^2*Kx.^2/4-lcy^2*Ky.^2/4);
    else
        W = (rmsh^2*lcx*lcy)./(2*pi*(1+(Kx*lcx).^2+(Ky*lcy).^2).^(3/2));
    end
    gam = randn(Ns) + 1i*randn(Ns);
    W = gam.*sqrt(W);
    dkx = 2*pi/Lx;
    dky = 2*pi/Ly;
    h = Nx*Ny*sqrt(dkx*dky)*real(ifft2(W));
% one correlation length is less than the sample rate
elseif sum(undersamp) == 1
    shift = find(~undersamp)-1;
    Ns = circshift(Ns,shift);
    dd = circshift(dd,shift);
    Ls = circshift(Ls,shift);
    Lcs = circshift(Lcs,shift);
    kx = 2*pi*fftfreq(1/dd(1),Ns(1));
    K = repmat(kx,1,Ns(2));
    lc = Lcs(1);
    if typesw
        W = (rmsh^2*lc/2/sqrt(pi))*exp(-lc^2*K.^2/4);
    else
        W = (rmsh^2*lc)./(pi*(1+(K*lc).^2));
    end
        gam = randn(Ns) + 1i*randn(Ns);
    gam(1,:) = 0;
    W = gam.*sqrt(W);
    dk = 2*pi/Ls(1);
    h = (Ns(1)*sqrt(dk))*real(ifft(W,[],1));
    h = shiftdim(h,shift);
% both correlation lengths less than their sample rates
else
    h = rmsh*randn(Ny,Nx);
end
```

10.3 3D Random Volumes

A 3D rough volume represents random fluctuations in space, such as random dielectric material or density variations. The Gaussian and exponential correlation functions are

$$C(\Delta x, \Delta y, \Delta z) = \exp\left(-\frac{\Delta x^2}{l_{cx}^2} - \frac{\Delta y^2}{l_{cy}^2} - \frac{\Delta z^2}{l_{cz}^2}\right) \tag{10.17}$$

$$C(\Delta x, \Delta y, \Delta z) = \exp\left(-\sqrt{\frac{\Delta x^2}{l_{cx}^2} + \frac{\Delta y^2}{l_{cy}^2} + \frac{\Delta z^2}{l_{cz}^2}}\right) \tag{10.18}$$

The corresponding power spectral densities are

$$W(k_x, k_y, k_z) = \frac{\sigma^2 l_{cx} l_{cy} l_{cz}}{8\pi^{3/2}} \exp\left(-\frac{k_x^2 l_{cx}^2}{4} - \frac{k_y^2 l_{cy}^2}{4} - \frac{k_z^2 l_{cz}^2}{4}\right) \tag{10.19}$$

$$W(k_x, k_y, k_z) = \frac{\sigma^2 l_{cx} l_{cy} l_{cz}}{3\pi\left(1 + k_x^2 l_{cx}^2 + k_y^2 l_{cy}^2 + k_z^2 l_{cz}^2\right)^2} \tag{10.20}$$

The spectrum is then

$$F(\mathbf{k}) = \gamma(\mathbf{k})\sqrt{\delta k_x \delta k_y \delta k_z W(\mathbf{k})} \tag{10.21}$$

$$\delta k_x = \frac{2\pi}{L_x} \quad \delta k_y = \frac{2\pi}{L_y} \quad \delta k_z = \frac{2\pi}{L_z} \quad k = \sqrt{k_x^2 + k_y^2 + k_z^2} \tag{10.22}$$

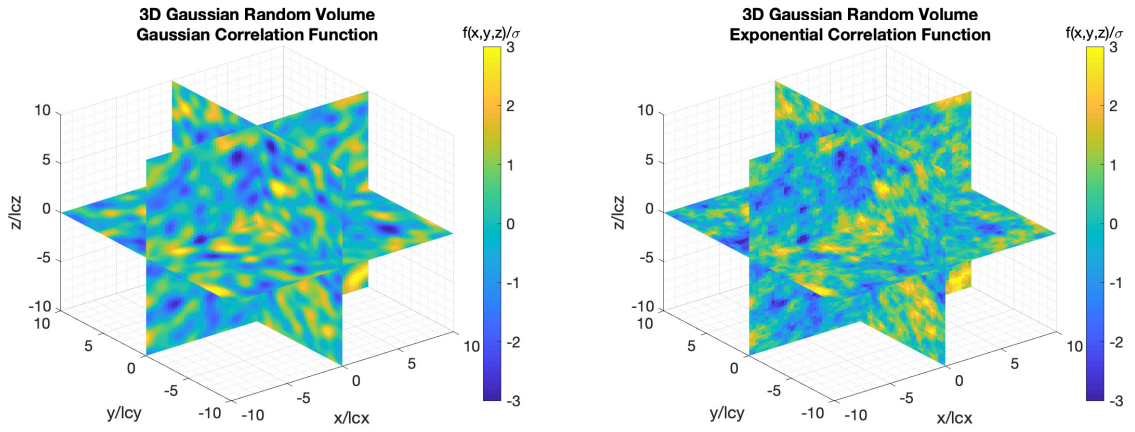


Figure 10.3: 3D Gaussian and exponential correlated random volumes generated with the same seed. The function value is normalized by the RMS, and the axes are normalized by the correlation length.

The routine `rough3` generates a real-valued zero-mean 3D Gaussian random volume. The inputs are the number of points N_x, N_y, N_z in each dimension, side lengths of the domain L_x, L_y, L_z, RMS fluctuation σ, and correlation lengths in each dimension l_{cx}, l_{cy}, l_{cz}. It returns the volume in `meshgrid` format. Use string switch `norm` or `exp` for Gaussian or exponential correlation functions. If the correlation lengths in all dimensions are less than their respective sample sizes, the values are independent draws from a Gaussian distribution. If correlation lengths in two dimensions are less than their sample sizes, the third dimension is composed of independent 1D random signals. If the correlation length of one dimension is less than its sample size, the other two dimensions are independent 2D random signals.

```
function [h] = rough3(Nx,Ny,Nz,Lx,Ly,Lz,rmsh,lcx,lcy,lcz,type)
% Generate 3D random volume
%
%   Nx,Ny,Nz:       number of points in x, y, and z
%   Lx,Ly,Lz:       physical extent of x, y, and z
%   rmsh:           root mean square height
%   lcx,lcy,lcz:    correlation length
%   type:           'norm' = Gaussian power spectral density
%                   'exp' = Exponential power spectral density
%
%   h:              [Ny,Nx,Nz] meshgrid array of values
%
% Dependencies: fftfreq

if strcmp(type,'norm')
    typesw = 1;
elseif strcmp(type,'exp')
    typesw = 0;
else
   error('bad PSD type')
end
dx = Lx/(Nx-1);
dy = Ly/(Ny-1);
dz = Lz/(Nz-1);
dd = [dy dx dz];
Ns = [Ny Nx Nz];
Ls = [Ly Lx Lz];
Lcs = [lcy lcx lcz];
undersamp = [(lcy<dy) (lcx<dx) (lcz<dz)];
% all correlation lengths are well sampled
if sum(undersamp) == 0
    kx = 2*pi*fftfreq(1/dx,Nx);
    ky = 2*pi*fftfreq(1/dy,Ny);
    kz = 2*pi*fftfreq(1/dz,Nz);
    [Kx Ky Kz] = meshgrid(kx,ky,kz);
    if typesw,
        W = (rmsh^2*lcx*lcy*lcz/8/(pi^(3/2)))*exp(-lcx^2*Kx.^2/4-lcy^2*Ky.^2/4-lcz^2*Kz.^2/4);
    else
        W = (rmsh^2*lcx*lcy*lcz)./(3*pi*(1+(Kx*lcx).^2+(Ky*lcy).^2+(Kz*lcz).^2).^2);
    end
    gam = randn(Ny,Nx,Nz) + 1i*randn(Ny,Nx,Nz);
    gam(1,1,1) = 0;
    W = gam.*sqrt(W);
    dkx = 2*pi/Lx;
    dky = 2*pi/Ly;
    dkz = 2*pi/Lz;
    h = (Nx*Ny*Nz*sqrt(dkx*dky*dkz))*real(ifftn(W));
% one correlation length is less than its sample rate
% shiftdim to put this in the last dimension
% use 2D PSD in dimensions 1 and 2, then shift back
elseif sum(undersamp) == 1
    dim = find(undersamp);
    shift = -dim;
    Ns = circshift(Ns,shift);
    dd = circshift(dd,shift);
    Ls = circshift(Ls,shift);
    Lcs = circshift(Lcs,shift);
    kx = 2*pi*fftfreq(1/dd(1),Ns(1));
    ky = 2*pi*fftfreq(1/dd(2),Ns(2));
    [Kx Ky] = meshgrid(kx,ky);
    Kx = repmat(Kx,1,1,Ns(3));
        Ky = repmat(Ky,1,1,Ns(3));
    lcx = Lcs(1);
    lcy = Lcs(2);
    if typesw
        W = (rmsh^2*lcx*lcy/4/pi)*exp(-lcx^2*Kx.^2/4-lcy^2*Ky.^2/4);
    else
        W = (rmsh^2*lcx*lcy)./(2*pi*(1+(Kx*lcx).^2+(Ky*lcy).^2).^(3/2));
```

```
    end
    gam = randn(Ns) + 1i*randn(Ns);
    gam(1,1,:) = 0;
    W = gam.*sqrt(W);
    dkx = 2*pi/Ls(1);
    dky = 2*pi/Ls(2);
    h = (Nx*Ny*sqrt(dkx*dky))*real(ifft(ifft(W,[],1),[],2));
    h = shiftdim(h,3+shift);
% two correlation lengths are less than their samples rates
% shiftdim to put one sampled dimension in dimension 1
% use 1D PSD in dimension 1, then shift back
elseif sum(undersamp) == 2
    dim = find(~undersamp);
    shift = 1-dim;
    Ns = circshift(Ns,shift);
    dd = circshift(dd,shift);
    Ls = circshift(Ls,shift);
    Lcs = circshift(Lcs,shift);
    kx = 2*pi*fftfreq(1/dd(1),Ns(1));
    K = repmat(kx,1,Ns(2),Ns(3));
    lc = Lcs(1);
    if typesw
        W = (rmsh^2*lc/2/sqrt(pi))*exp(-lc^2*K.^2/4);
    else
        W = (rmsh^2*lc)./(pi*(1+(K*lc).^2));
    end
    gam = randn(Ns) + 1i*randn(Ns);
    gam(1,:,:) = 0;
    W = gam.*sqrt(W);
    dk = 2*pi/Ls(1);
    h = (Ns(1)*sqrt(dk))*real(ifft(W,[],1));
    h = shiftdim(h,3+shift);
% all correlation lengths less than sample rate
else
    h = rmsh*randn(Ny,Nx,Nz);
end
```

10.4 1D Fractal Signals

Fractal signals are a class of Gaussian random process that are self-similar at increasing length scales. In short, the RMS grows as the profile length grows. Equivalently, these signals describe fractional Brownian motion, of which traditional Brownian motion is a subset. For in-depth explanations of fractal surfaces, theory, and relation to natural surfaces, see [54, 55]. For details on computing these signals quickly, see [56].

The statistics of 1D fractal signals are described by either an RMS height (or variance) over all points which depends on total length of the profile, or the RMS deviation (or Allan variance), that depends on the relative positions, or lag, between pairs of points. For a 1D profile with height $z(x)$, the variance of the heights is computed as

$$\sigma^2 = \left\langle (z - \bar{z})^2 \right\rangle \tag{10.23}$$

and the Allan variance is computed as

$$v^2 = \left\langle (z(x) - z(x + \Delta x))^2 \right\rangle \tag{10.24}$$

where Δx is the lag and $\langle\rangle$ is the ensemble average. For natural fractal surfaces, these quantities can be described by a power law such that the RMS height over the entire profile length, x, is

$$\sigma(x) = \sigma_o \left(\frac{x}{x_o} \right)^H \tag{10.25}$$

where σ_o is the RMS height at length scale x_o, and H is the Hurst exponent, which takes values $0 \leq H \leq 1$. Alternatively, the surface can be described by a power law such that the RMS deviation (Allan deviation, or just deviation) at lag Δx is given by

$$v(\Delta x) = v_o \left(\frac{\Delta x}{\Delta x_o} \right)^H \tag{10.26}$$

where v_o is the RMS deviation at lag Δx_o, and Δx is the lag at which we evaluate the deviation. The Hurst exponent controls the growth of the surface over distance, while the other parameters simply scale the result. The statistics of both the surface height and variation are Gaussian. When $H = 1/2$ the process becomes standard 1D Brownian motion.

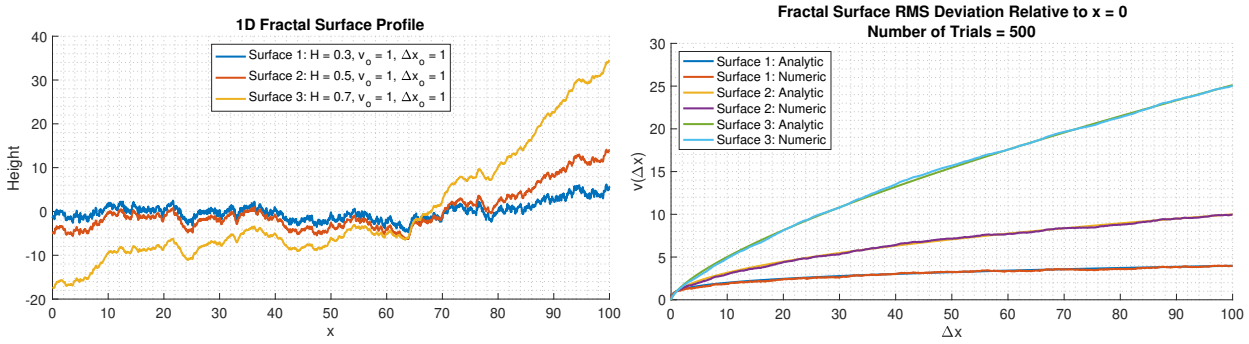


Figure 10.4: Top: 1D fractal profiles generated from the same seed. Bottom: RMS deviation relative to $x = 0$, $v(\Delta x)$, analytic and numeric computed over 500 trials. This shows how the deviation of the fractal surface continues to grow with length scale.

From [54], the RMS slope is defined

$$s_{rms} = \sqrt{\left\langle \left(\frac{\Delta z}{\Delta x} \right)^2 \right\rangle} \tag{10.27}$$

The RMS of Δz is just (10.26), therefore, the RMS slope is

$$s_{rms} = \frac{v(\Delta x)}{\Delta x} = v_o \left(\frac{\Delta x}{\Delta x_o} \right)^{H-1} \tag{10.28}$$

The routine `fractal1` generates a zero-mean 1D fractal signal based on the routine in [56], to which we have added options for normalization by RMS height or RMS deviation. It takes as input the pair (σ_o, x_o) or (v_o, Δx_o), the Hurst exponent H, the number of sample points, and the length of the profile. Use string switch `'rms'` or `'dev'` for normalization by RMS height or RMS deviation. The signal generated by [56] was found to scale perfectly for RMS deviation, (10.26). For RMS height, the signal is simply rescaled by (10.25).

```
function z = fractal1(Nx,Lx,H,sig_o,x_o,type)
% 1D Fractal Brownian profile
%
% Nx:       Number of points in the profile
% Lx:       Profile length
% H:        Hurst exponent between [0,1];
% sig_o:    RMS height \sigma_o at scale x_o (or RMS deviation v_o at lag \Delta x_o)
% x_o:      Length scale x_o at which sig_o applies (or lag \Delta x_o at which v_o applies)
% type:     'rms' for RMS height (sig_o, x_o)
%           'dev' for RMS deviation (v_o, \Delta x_o)
%
% z:        1D height profile
%
% Notes: Based on fbm1d.m (Kroese, et. al, 2015)

if H < 0 || H > 1
    error('Bad Hurst exponent')
end
if strcmp(type,'rms')
    typein = 1;
elseif strcmp(type,'dev')
    typein = 0;
else
    error('Bad type')
end
% covariance
R = nan(Nx+1,1);
R(1) = 1;
ind = 1:Nx;
R(ind+1) = 0.5*((ind+1).^(2*H) - 2*ind.^(2*H) + (ind-1).^(2*H));
R = [R; R(end-1:-1:2)]; % load the circulant matrix
lambda = real(fft(R))/(2*Nx); % eigenvalues
gam = randn(2*Nx,1) + 1i*randn(2*Nx,1); % random seed
z = real(fft(gam.*sqrt(lambda)));
z = Nx^(-H)*cumsum(z(1:Nx+1));
z = z(1:Nx);
z = z-mean(z); % zero mean
const = sig_o*(Lx/x_o)^H;
% normalize according to RMS height or RMS variation
if typein
    z = const*z/std(z);
else
    z = const*z;
end
```

10.5 Bicontinuous Random Media

Bicontinuous random media are used to model 2-species material with different levels of connectivity. For example, these have been used to model the distribution of ice/void volumes of settled snow that have different amounts of melting and compaction, [57, 58, 59, 60]. Bicontinuous random media partition the volume into binary regions by cutting a fluctuating 3D field at a prescribed level and then assigning the material based on whether the value of the field is above or below the level cut at a given point. The field has Gaussian statistics, therefore the cutting level corresponds to the volume fraction of the binary regions. We outline the components of the model given in [57, 58] for isotropic bicontinuous random media, while an anisotropic model can be found in [60].

Distribution The 3D fluctuating field is a sum of random plane waves

$$S(\mathbf{r}) = \frac{1}{\sqrt{N}} \sum_{n=1}^{N} \cos\left(\mathbf{k}_n \cdot \mathbf{r} + \phi_n\right) \tag{10.29}$$

where $S(\mathbf{r})$ is the fluctuating field, N is the number of wave vector directions, $\mathbf{k}_n$ is the wave vector, and ϕ_n is a random phase. The wave vector directions, $\hat{k}_n$, are distributed uniformly random over the sphere. The wave vector magnitudes, k_n, are drawn from a gamma distribution. The phase is drawn from a uniform random variable between 0 and 2π. The gamma distribution for the wave vector amplitudes is given by, [57],

$$p(k) = \frac{1}{\Gamma(b+1)} \frac{(b+1)^{b+1}}{\langle k \rangle} \left(\frac{k}{\langle k \rangle}\right)^{b} e^{-(b+1)\frac{k}{\langle k \rangle}} \tag{10.30}$$

where Γ is the gamma function, $\langle k \rangle$ is the mean wavenumber and b is a constant. Physically, $\langle k \rangle$ corresponds to the reciprocal of the average length scale of heterogeneity (average snow grain size), while b corresponds to the width of the distribution (spread of snow grain size) [58]. Despite its appearance, (10.30) is equivalent to the common form of the gamma distribution that has shape parameter $b+1$ and scale parameter $\langle k \rangle / (b+1)$. These parameters are needed to draw samples from built-in routines. The mean of the distribution is $\langle k \rangle$ and the standard deviation is $\sigma = \langle k \rangle / \sqrt{b+1}$.

An indictor function is used to assign the binary classification to each point in the field based on a level cut. The indicator function is

$$\Theta(S(\mathbf{r})) = \begin{cases} 1, & S(\mathbf{r}) > \alpha \\ 0, & S(\mathbf{r}) \leq \alpha \end{cases} \tag{10.31}$$

where $\Theta(\mathbf{r})$ is the indicator function and α is the cutting level. The statistics of the field are zero-mean Gaussian with variance 1/2. Because of this, the fractional volume, f_v, of the 1 indicator is related to the cutting level as

$$f_v = \frac{1}{2}\left(1 - \mathrm{erf}(\alpha)\right) \tag{10.32}$$

where erf is the error function. The cutting level is given in terms of the fractional volume as

$$\alpha = \mathrm{erf}^{-1}\left(1 - 2f_v\right) \tag{10.33}$$

where erf^{-1} is the inverse error function.

The construction of (10.29) as a sum over spatial plane waves has several interesting consequences. The direct sum, together with the fact that N needs to be large, makes the computation slow over a large number of points. In [57] a value of $N = 10^4$ is used, though we have found that $N = 10^3$ suffices. Also, because this does not use a PSD, FFT methods cannot be used to accelerate the computation. However, precisely because (10.29) is spatially sampled and not based on a PSD or FFT, the 3D field and indicator function can be computed at arbitrary points, for example, along 1D lines, 2D sheets, or 3D volumes. This allows large volumes to be computed in pieces by using the same random seed.

Correlation Function The spatial correlation function of $\Theta(\mathbf{r})$ is radially symmetric and given by, [57],

$$\Gamma_\alpha(r) = f_v^2 + C_\alpha(r) \tag{10.34}$$

where

$$C_\alpha(r) = \sum_{m=1}^{\infty} C_m(\alpha)\left[C_s(r)\right]^m \tag{10.35}$$

$$C_m(\alpha) = \frac{e^{-2\alpha^2}\left[H_{m-1}(\alpha)\right]^2}{\pi m! 2^m} \tag{10.36}$$

$$C_s(r) = \frac{\mathrm{sinc}(b\varphi)}{\mathrm{sinc}(\varphi)}\cos^{b+1}(\varphi) \tag{10.37}$$

$$\varphi = \tan^{-1}\left(\frac{\langle k\rangle\, r}{b+1}\right) \tag{10.38}$$

where $C_\alpha(r) = \Gamma_\alpha(r) - f_v^2$ and $C_s(r)$ is the autocovariance of $S(\mathbf{r})$. The coefficients, $C_m(\alpha)$, are given in terms of the Hermite polynomials, $H_m(x)$. In addition, $\mathrm{sinc}(x) = \sin(x)/x$. Finally, $\Gamma_\alpha(0) = f_v$, $\Gamma_\alpha(\infty) = f_v^2$, $C_s(0) = 1$, and $C_s(\infty) = 0$.

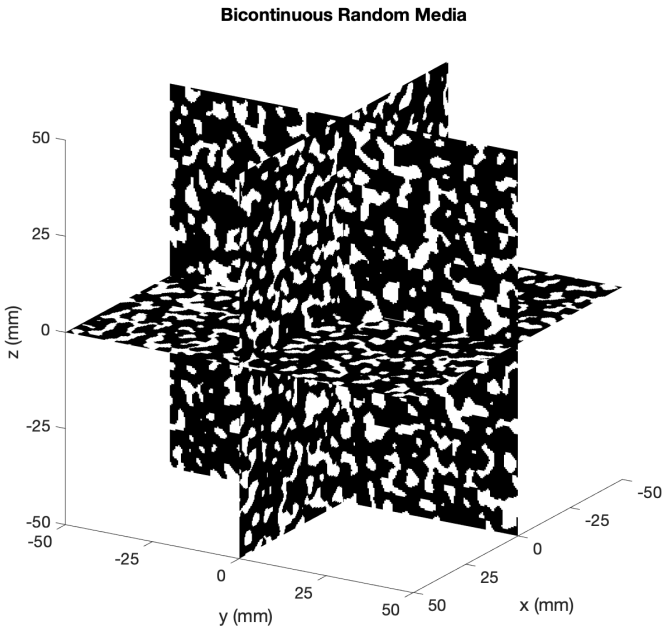


Figure 10.5: Bicontinuous random media for $l_{ave} = 2\pi/\langle k\rangle = 5$ mm, $b = 10$, and $f_v = 0.3$. The 2D slices of the same field are computed separately by running the routine again with the same random seed.

Routine The routine `bicont` returns the bicontinuous indicator function, $\Theta(\mathbf{r})$. It takes as input the arrays of the Cartesian coordinates $\mathbf{r} = (x, y, z)$, which can be any size, the mean length scale of the heterogeneity, l_{ave}, spreading parameter, b, and fractional volume, f_v. The mean length scale is converted to mean wavenumber as $\langle k\rangle = 2\pi/l_{ave}$. l_{ave} needs to have the same units as the coordinates. The number of plane waves, N, is optional and defaults to $N = 10^3$. Random plane directions are computed with our routine `randsphere`. This relies on Matlab routines `gamrnd` and `erfinv` which require certain toolboxes. The fluctuating field, $S(\mathbf{r})$, is an optional output.

```
function [Theta S] = bicont(X,Y,Z,lave,b,fv,N)
% Bicontinuous random media
%
% X,Y,Z     [any size] Cartesian sample points
% lave      Average length scale of heterogeneity
% b         Shape parameter
% fv        Volume fraction between 0 and 1
% N         [optional] Number of plane waves
%
% Theta     Bicontinuous indicator function
% S         [optional] Fluctuating field
%
% Dependencies: randsphere,sph2cart

if nargin == 6
    N = 10^3;
end

% scale and shape parameters
kave = 2*pi/lave;
shape = b+1;
scale = kave/(b+1);
alp = erfinv(1-2*fv);

% compute random wave vectors
k_r = gamrnd(shape,scale,N,1);
[k_th k_ph] = randsphere(N);
[kx ky kz] = sph2cart(k_r,k_th,k_ph);
phi = 2*pi*rand(N,1);

% create fluctuating field
S = zeros(size(X));
for n=1:N
    S = S + cos(kx(n)*X + ky(n)*Y + kz(n)*Z + phi(n));
end
S = S/sqrt(N);

% indicator function from level cut
Theta = zeros(size(S));
Theta(S > alp) = 1;
```

10.6 Ocean Waves

Ocean wave scenes are generated as a 2D rough surface. Ocean waves have a nonlinear dispersion relation, so some care is needed when deriving the power spectral density. Models for temporal evolution also come in linear and nonlinear flavors, but all use the same underlying PSD. Note, only one realization of the ocean spectrum is needed in order to evolve the surface in time.

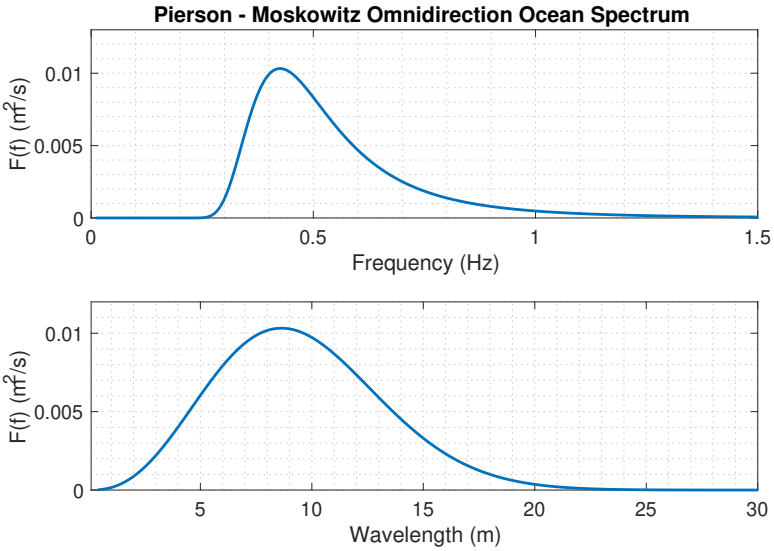


Figure 10.6: Pierson-Moskowitz spectrum for $U_{10} = 3$ m/s

The classic Pierson-Moskowitz (PM) omnidirectional wave spectrum for fully developed ocean waves is

$$F(f) = \frac{\alpha g^2}{(2\pi)^4 f^5} \exp\left[-(5/4)\left(\frac{f_p}{f}\right)^4\right] \tag{10.39}$$

where $\alpha = 0.0081$ is the Phillips constant, $g = 9.81$ m/s^2 is gravitational acceleration, f is the spectral frequency, and f_p is the peak frequency, both frequencies have units of Hz [61]. This is an omnidirectional spectrum; angular variation will be added later to account for wind direction. The peak frequency is given empirically as

$$f_p = 0.13\frac{g}{U_{10}} \tag{10.40}$$

where U_{10} is the wind speed 10 meters above the mean surface level. The variance of the wave height is given by the integral over the density function

$$\sigma^2 = \int_0^\infty F(f)df = \frac{\alpha g^2}{(2\pi)^4 5 f_p^4} \tag{10.41}$$

The RMS wave height is then

$$\sigma = \frac{\sqrt{\alpha} g}{(2\pi)^2 \sqrt{5} f_p^2} \tag{10.42}$$

The dispersion relation for ocean waves is $\omega^2 = gk$ from which it follows that the phase velocity, group velocity, and wavelength are

$$v_p = \frac{\omega}{k} = \sqrt{\frac{g}{k}} \tag{10.43}$$

$$v_g = \frac{d\omega}{dk} = \frac{g}{2\omega} \tag{10.44}$$

$$\lambda = \frac{g}{2\pi f^2} \tag{10.45}$$

To create wave scenes, the omnidirectional spectrum given in frequency needs to be converted to 2D k-space. This requires a change of variables. If one desires the 1D ocean spectra in k-space, the change of variables via the Jacobian is explained in [62]. How to convert to the 2D omnidirectional spectra is explained in [63] and repeated here. The total wave height variance integrated over all angles must not depend on the independent variable, f or k. This condition is met when

$$\int F(k)kdk = \int F(f)df \tag{10.46}$$

which yields the change of variables

$$F(k) = F(f)\frac{df}{kdk} = F(f)\frac{d\omega}{2\pi kdk} = F(f)\frac{v_g}{2\pi k} \tag{10.47}$$

where $F(k)$ is the 2D omnidirectional spectrum. Next, the 2D PSD is the product of the omnidirectional spectrum and an angle dependent spreading function [64]:

$$W(\mathbf{k}) = F(k)\phi(\theta) \tag{10.48}$$

where $\mathbf{k}$ is the 2D wavenumber and (k, θ) are its polar coordinates. The spreading function can take a few forms, most are some power of $\cos(\theta)$, given in [64] as

$$\phi(\theta) = \frac{1}{N}\left|\cos^5\left(\frac{\theta - \theta_v + \pi}{2}\right)\right| \tag{10.49}$$

where θ_v is the direction from which the wind is blowing (meteorological convention). The normalization is $N = \int_{-\pi}^{\pi} |\cos^5(\theta/2)|d\theta = 32/15$. More general spreading functions are given in [65, 66]. A factor of π has been added to the argument to give the spectrum the correct orientation relative to our definition of the wind direction, the k-vectors, and FFT convention.

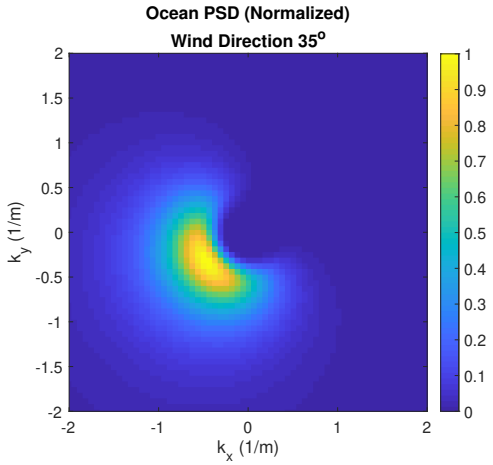


Figure 10.7: 2D ocean PSD (normalized) for $U_{10} = 3$ m/s, and $\theta_v = 35^o$. Most waves travel downwind, some have an upwind component, none travel directly upwind.

From [64, 66], the complex amplitude of the wave heights at time t is given by

$$A(\mathbf{k}, t) = \gamma(\mathbf{k})\sqrt{\delta k_x \delta k_y W(\mathbf{k})}e^{-i\omega t} \tag{10.50}$$

where $\gamma(\mathbf{k})$ is the complex standard Gaussian distribution. ω is replaced with the dispersion relation, $\sqrt{gk}$, giving a spectrum in terms of k and t. The effect of the nonlinear dispersion relation is that different parts of the spatial spectrum oscillate at different rates in time. The steps are $\delta k_x = 2\pi/L_x$ and $\delta k_y = 2\pi/L_y$ where L_x and L_y and the lengths for domain in each dimension.

For a linear sea surface, the surface height is the real part of the 2D inverse Fourier transform of the complex amplitudes.

$$A(\mathbf{x},t) = \mathrm{Re}\mathcal{F}_{\mathbf{k}}^{-1}\left[A(\mathbf{k},t)\right] \tag{10.51}$$

A collection of nonlinear ocean wave models can be found in [67]. A simple nonlinear model is the Creamer 2 surface, [64]. It adds a term to the spectrum, which modifies the features of the peaks and troughs slightly. The additional spectral term is

$$C_t^2(\mathbf{k}) = -\frac{k_x^2}{2k}\mathcal{F}\left[h_{t_x}^2\right] - \frac{k_x k_y}{k}\mathcal{F}\left[h_{t_x}h_{t_y}\right] - \frac{k_y^2}{2k}\mathcal{F}\left[h_{t_y}^2\right] \tag{10.52}$$

where h_{t_x} and h_{t_y} are the components of the Hilbert transform of the surface

$$\mathbf{h}_t(\mathbf{x}) = \mathrm{Re}\sum_{\mathbf{k}}\left(-i\frac{\mathbf{k}}{k}\right)A_t(\mathbf{k})e^{i\mathbf{k}\cdot\mathbf{x}} \tag{10.53}$$

The Hilbert transform is computed in the spatial frequency domain. Numerically, the infinite values that result from divide by $k = 0$ are zeroed out, and a wide radial low pass filer is applied to $C_t^2(\mathbf{k})$ to clean the spectrum before the last transform. The filter used is

$$B(k) = \frac{1}{1+\left(\frac{k}{k_c}\right)^{\mu}} \tag{10.54}$$

where k_c is the cutoff, and μ controls the roll off. Values of $k_c = 5$ and $\mu = 8$ have been used. The final surface is the sum of linear and nonlinear surfaces

$$A_t(\mathbf{x}) = \mathrm{Re}\mathcal{F}^{-1}\left[A_t(\mathbf{k}) + B(k)C_t^2(\mathbf{k})\right] \tag{10.55}$$

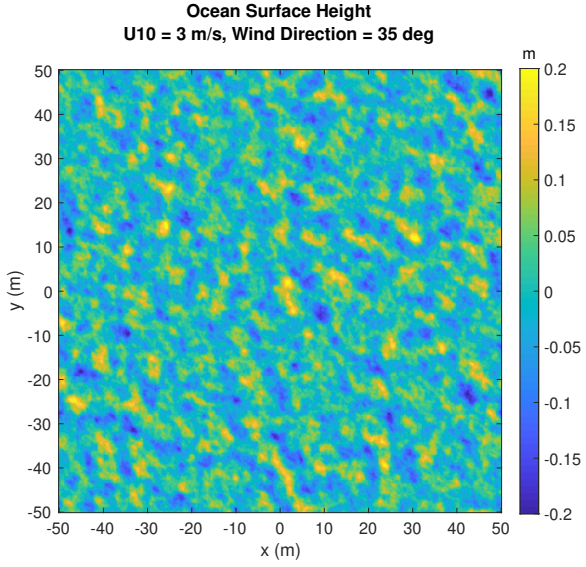


Figure 10.8: Simulated wave heights at $t = 0$ for $U_{10} = 3$ m/s, and $\theta_{\mathrm{v}} = 35^o$.

The function `psdOcean` produces the 2D ocean wave PSD as a combination of the Pierson-Moskowitz 1D omnidirectional spectrum and a wind direction spreading function. It also returns the k-space components which are needed to evolve the surface. The routine `evolveOcean` creates the surface at one instance in time, taking as inputs the ocean PSD and wavenumber components from `psdOcean` and one realization of γ. Use the string switch `creamer2` to activate the nonlinear term. Numerically, the RMS wave height of the generated ocean surface does not always match the analytical value, therefore, consider renormalizing the surface RMS to the value given by (10.42).

```
function [W, Kx, Ky] = psdOcean(Nx,Ny,Lx,Ly,u10,wind_dir_deg)
% Generate power spectral density of ocean waves based on
% Pierson - Moskowitz (PM) wave spectrum
%
% Nx, Ny:         Number of points in x and y
% Lx, Ly:         Domain length in x and y
% u10:            Wind speed 10 meters above mean sea height
% wind_dir_deg:   Direction from which wind blows, in degrees
%                (meteorological convention)
%
% W:              power spectral density, 2D unshifted FFT format
% Kx, Ky:         2D unshifted FFT and meshgrid format
%
% Dependencies: fftfreq

g = 9.81;                       % gravitational acceleration, (m/s^2)
alpha = 0.0081;                 % Phillips constant
wind_dir_rad = pi/180*wind_dir_deg;
fp = 0.13*g/u10;              % PM peak freqeucy (Hz)
const_f = alpha*g^2*(2*pi)^-4;  % PM scale constant
dx = Lx/(Nx-1);
dy = Ly/(Ny-1);
kx = 2*pi*fftfreq(1/dx,Nx);
ky = 2*pi*fftfreq(1/dy,Ny);
[Kx, Ky] = meshgrid(kx,ky);
K = sqrt(Kx.^2 + Ky.^2);
Theta = atan2(Ky,Kx);
F = sqrt(g*K)/2/pi;     % frequency on k-space grid, from dispersion relation
vg = g./(2*2*pi*F);     % group velocity
cv = vg./(2*pi*K);      % change of variables
Fk = cv.*const_f.*(F.^-5).*exp(-(5/4)*(fp./F).^4); % PM spectrum, k-space
Fk(find(K == 0)) = 0;        % zero mean
phi = (32/15)*abs(cos(0.5*(Theta-wind_dir_rad+pi)).^5);
W = Fk.*phi;

function At = evolveOcean(W,Kx,Ky,gam,t,type)
% Evolve an ocean surface at time t given the spectrum and time
% Creamer2 nonlinear surface is an option.
%
% W:            Ocean spectrum from psdOcean
% gam:          One realization of 2D complex standard normal
% Kx,Ky:        Spatial frequency matrices from psdOcean
% t:            Time (s)
% type:         'creamer2' for nonlinear surface
%
% At:           Ocean surface at time t
%
% Dependencies: psdOcean

g = 9.81;           % gravitational acceleration, (m/s^2)
K = sqrt(Kx.^2 + Ky.^2);
ww = sqrt(g*K);     % dispersion relation
dkx = Kx(1,2) - Kx(1,1);
dky = Ky(2,1) - Ky(1,1);
Atk = gam.*sqrt(dkx*dky*W).*exp(-1i*ww*t); % linear complex amplitudes
[Nx Ny] = size(W);
At = (Nx*Ny)*real(ifft2(Atk)); % linear surface height
if nargin == 6 && strcmp(type,'creamer2')
   % Hilbert Transform
   htk_x = -1i*(Kx./K).*Atk;
   htk_y = -1i*(Ky./K).*Atk;
   indInf = find(K == 0);
   htk_x(indInf) = 0;
   htk_y(indInf) = 0;
   htx_x = real(ifft2(htk_x));
   htx_y = real(ifft2(htk_y));
   % Creamer 2 spectrum
   Ct2k = -(Kx.^2)./(2*K).*fft2(htx_x.^2) ...
         -(Kx.*Ky)./(K).*fft2(htx_x.*htx_y) ...
         -(Ky.^2)./(2*K).*fft2(htx_y.^2);
   Ct2k(indInf) = 0;
   % Radial lowpass filter
   kcutoff = 5;
   mu = 8;
   Bk = 1./(1+(K/kcutoff).^(mu));
   At = Nx*Ny*real(ifft2(Atk + Bk.*Ct2k));
end
```

10.7 Gaussian Random Particles

In this section we give a procedure for creating and computing a spherical particle with Gaussian surface roughness. This is based on the formulation in [68]. Specifically, the radius of the particle surface has log-normal statistics and the radii are correlated through a circular isotropic Gaussian angular correlation function. We have added 1) a derivation for the spherical harmonic expansion coefficients of the angular correlation function, which was not included in the paper, 2) computation using normalized Legendre polynomials, as well as 3) a stable way to carry out the computation in the small correlation limit.

Log-normal Radius From [68], the particle radius at spherical coordinate (θ, ϕ) is described by the log-normal distribution

$$r(\theta, \phi) = \frac{a}{\sqrt{1+\sigma^2}} \exp\left[s(\theta, \phi)\right] \tag{10.56}$$

which has mean a, and variance $a^2\sigma^2$, where $s(\theta, \phi)$ is a zero-mean Gaussian random variable at each point (θ, ϕ) with variance β^2. The covariance of s between two spherical points (θ_i, ϕ_i), (θ_j, ϕ_j) is given by

$$\Sigma_{s,ij} = \beta^2 C_s(\gamma_{ij}) \tag{10.57}$$

where γ_{ij} is the angular correlation between directions i and j and C_s is the correlation function. The covariances and distribution parameters are related as

$$\begin{aligned} \Sigma_{r,ij} &= a^2\left[\exp(\Sigma_{s,ij}) - 1\right] && (10.58)\\ \sigma^2 C_r &= \exp(\beta^2 C_s) - 1 && (10.59)\\ \sigma^2 &= \exp(\beta^2) - 1 && (10.60) \end{aligned}$$

Spherical Harmonic Expansion The random variable, s, and angular correlation function are expressed in terms of real-valued spherical harmonics and associated Legendre polynomials, respectively, as

$$s(\theta, \phi) = \sum_{l=0}^{\infty} \sum_{m=0}^{l} P_l^m(\cos\theta)(a_{lm}\cos(m\phi) + b_{lm}\sin(m\phi)) \tag{10.61}$$

$$C_s(\gamma) = \sum_{l=0}^{\infty} c_l P_l(\cos\gamma) \tag{10.62}$$

where a_{lm} and b_{lm} are zero-mean Gaussian random variables with variance

$$\beta_{lm}^2 = (2 - \delta_{m,0})\frac{(l-m)!}{(l+m)!}\beta^2 c_l \tag{10.63}$$

The correlation function is chosen as a spherical Gaussian of the form:

$$\begin{aligned} C_s(\gamma) &= \exp\left(-\frac{2}{l_c^2}\sin^2(\gamma/2)\right) && (10.64)\\ l_c &= 2\sin(\Gamma/2) && (10.65) \end{aligned}$$

where l_c is the angular correlation (unitless) and Γ is the correlation angle (radians). The function is circular such that the derivative at $\gamma = 0$ and $\gamma = \pi$ is zero. The maximum correlation angle is $\Gamma = \pi$, for which the value at the maximum angle of separation, $\gamma = \pi$, is $\exp(-1/2) \approx 0.61$. Using (10.62), (10.64) and orthogonality (derivation below) the expansion coefficients for the correlation function are

$$c_l = (2l+1)\exp\left(-\frac{1}{l_c^2}\right) i_l\left(\frac{1}{l_c^2}\right) \tag{10.66}$$

where $i_l(x)$ is the modified spherical Bessel function of the first kind. The number of required harmonics in the expansion increases as the correlation length decreases. [68] uses the following heuristic for 5 digits of precision:

$$L_{max} = \frac{275^o}{\Gamma} + 2.5 \tag{10.67}$$

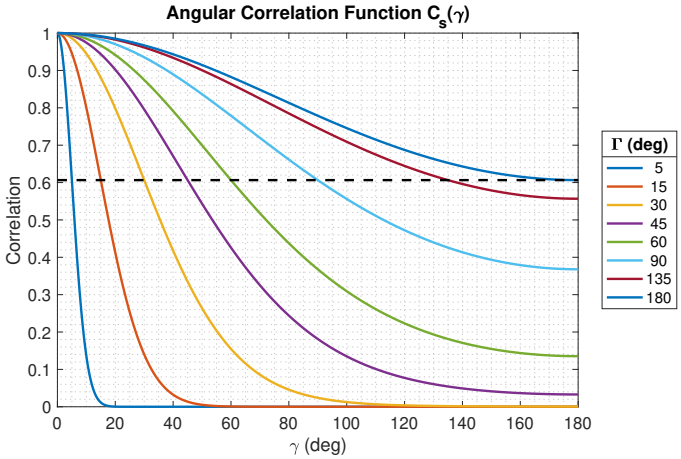


Figure 10.9: Angular correlation function, $C_s(\gamma)$.

Derivation of c_l A derivation for c_l was not included in [68]. Applying orthogonality of the Legendre polynomials in (10.62), the expansion coefficients of the correlation function are generally

$$c_l = \frac{2l+1}{2}\int_0^\pi C_s(\gamma)P_l(\cos\gamma)\sin\gamma d\gamma \tag{10.68}$$

Using $\sin^2(\gamma/2) = (1-\cos\gamma)/2$, the correlation function can be written

$$C_s(\gamma) = \exp\left(-\frac{1}{l_c^2}(1-\cos\gamma)\right) = \exp\left(-\frac{1}{l_c^2}\right)\exp\left(\frac{1}{l_c^2}\cos\gamma\right) \tag{10.69}$$

after which the integral becomes

$$c_l = \frac{2l+1}{2}\exp\left(-\frac{1}{l_c^2}\right)\int_0^\pi \exp\left(\frac{1}{l_c^2}\cos\gamma\right)P_l(\cos\gamma)\sin\gamma d\gamma \tag{10.70}$$

From [69, 70], this integral has the form

$$\int_0^\pi \exp\left[\pm iR\cos\theta\right]P_l^{|m|}(\cos\theta)\sin^{|m|+1}\theta d\theta = 2(\pm i)^{l+|m|}\frac{(l+|m|)!}{(l-|m|)!}\frac{j_l(R)}{R^{|m|}} \tag{10.71}$$

Using $m=0$ and identifying $-iR = 1/l_c^2$, then (10.70) reduces to

$$c_l = \frac{2l+1}{2}\exp\left(-\frac{1}{l_c^2}\right)2(-i)^l j_l\left(i\frac{1}{l_c^2}\right) \tag{10.72}$$

Using $(-i)^n = i^{-n}$, and the fact that $i_n(x) = i^{-n}j_n(ix)$, where $i_n(x)$ is the modified spherical Bessel function of the first kind,

$$i_n(x) = \sqrt{\frac{\pi}{2x}}I_{n+1/2}(x) \tag{10.73}$$

the expansion coefficients for the correlation function simplify to

$$c_l = (2l+1)\exp\left(-\frac{1}{l_c^2}\right)i_l\left(\frac{1}{l_c^2}\right) \tag{10.74}$$

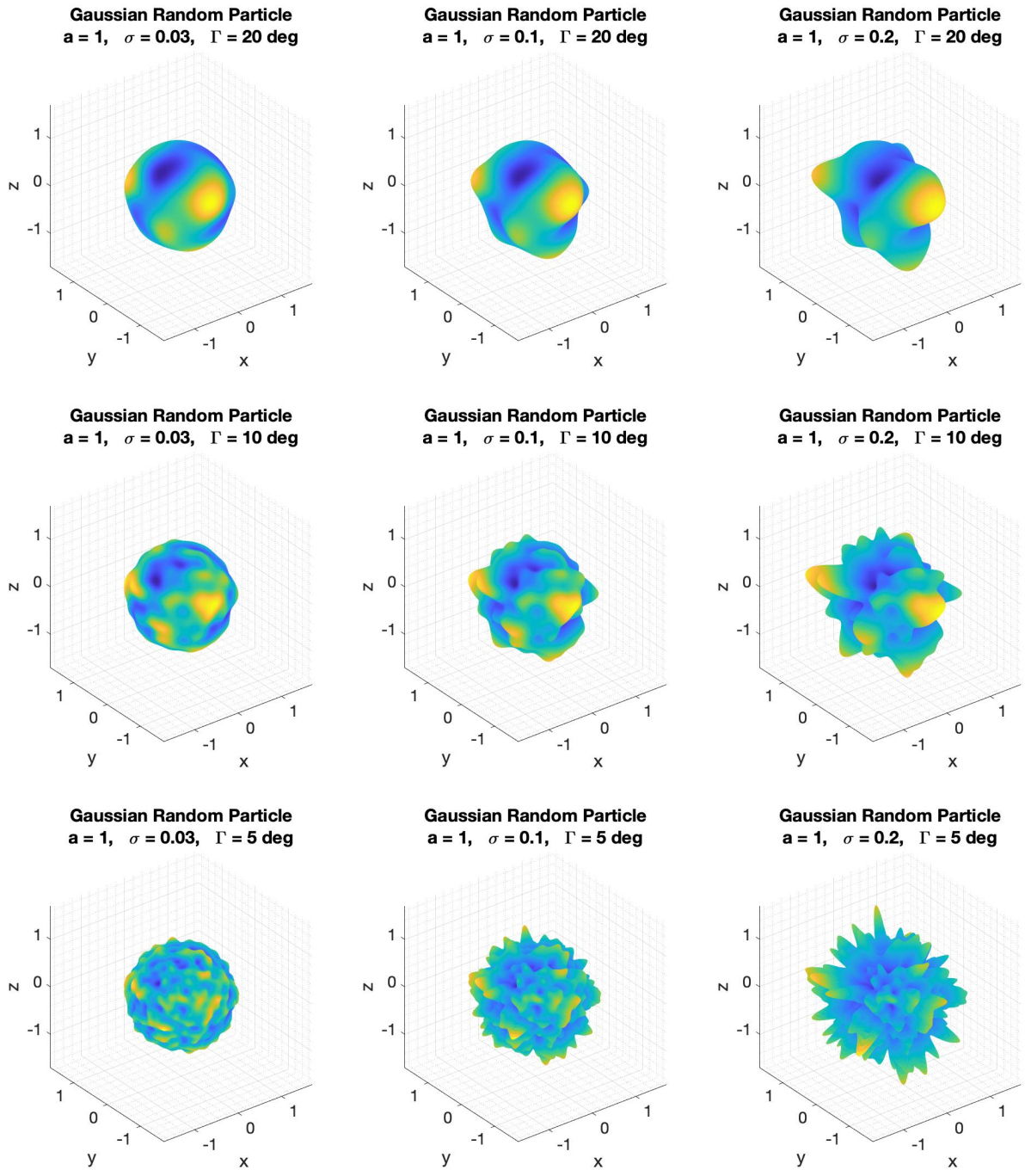


Figure 10.10: Examples of Gaussian random particles with mean radius, $a = 1$, for different log-normal RMS, $a\sigma$, and angular correlations, Γ. Each correlation value requires a different number of total harmonics, however the spherical harmonics coefficients are computed from the same random seed (i.e., the random draws count up from the lowest harmonic), which ensures that the same features are observed across the different particles.

Fully Normalized Legendre Polynomials Equation (10.61) is best computed with the fully normalized Legendre polynomials, which avoid direct computation of the factorials. Defining a'_{lm} and b'_{lm} as draws from standard normal distribution with zero mean and unit variance, (10.61) can be written

$$s(\theta,\phi) = \sum_{l=0}^{\infty}\sum_{m=0}^{l} P_l^m(\cos\theta)\sqrt{(2-\delta_{m,0})\frac{(l-m)!}{(l+m)!}}c_l\beta(a'_{lm}\cos(m\phi)+b'_{lm}\sin(m\phi)) \tag{10.75}$$

The fully normalized associated Legendre polynomials are

$$\widetilde{P}_l^m(x) = \sqrt{\frac{(l+1/2)(l-m)!}{(l+m)!}}P_l^m(x) \tag{10.76}$$

Substituting (10.74) into (10.75) and using (10.76), we get

$$s(\theta,\phi) = \beta\sum_{l=0}^{\infty}\sum_{m=0}^{l} c'_{lm}\widetilde{P}_l^m(\cos\theta)(a'_{lm}\cos(m\phi)+b'_{lm}\sin(m\phi)) \tag{10.77}$$

where

$$c'_{lm} = \sqrt{2(2-\delta_{m,0})\exp\left(-\frac{1}{l_c^2}\right)i_l\left(\frac{1}{l_c^2}\right)} \tag{10.78}$$

Small Correlation Limit When the correlation angle is small, e.g., $\Gamma < 3^o$, the following computation is problematic

$$A_l(l_c) = \exp\left(-\frac{1}{l_c^2}\right)i_l\left(\frac{1}{l_c^2}\right) \tag{10.79}$$

because the exponent goes to zero and the Bessel function becomes large. The solution is to use the log transform as well as recursion over the series expansion of the Bessel function. From [71, Eq. 10.53.3], the series expansion of the modified Bessel function is

$$i_l(z) = z^l\sum_{k=0}^{\infty}\frac{(1/2z^2)^k}{k!(2l+2k+1)!!} \tag{10.80}$$

Setting $z = 1/l_c^2$ and pulling all terms into the sum (10.79) can be written

$$A_l(z) = \sum_{k=0}^{\infty} e^{-z}z^l\frac{(1/2z^2)^k}{k!(2l+2k+1)!!} = \sum_{k=0}^{\infty} a_{l,k} \tag{10.81}$$

Using the log transform, the terms of the sum are expressed as $a_{l,k} = e^{\ln a_{l,k}}$. Writing out the logarithm

$$\ln a_{l,k} = -z + l\ln z + k(-\ln 2 + 2\ln z) - \ln k! - \ln(2l+2k+1)!! \tag{10.82}$$

The double factorial for odd integers is $n!! = \prod_{k=1}^{(n+1)/2}(2k-1)$. Using this in (10.82) and converting the products to sums

$$\ln a_{l,k} = -z + l\ln z + k(-\ln 2 + 2\ln z) - \sum_{n=1}^{k}\ln n - \sum_{n=1}^{l+k+1}\ln(2n-1) \tag{10.83}$$

To avoid recomputing the sums for new l and k, fast recursion relations are derived by writing out $l+1$ or $k+1$ and separating $\ln a_{l,k}$:

$$\ln a_{l+1,k} = \ln a_{l,k} + \ln z - \ln(2(l+k)+3) \tag{10.84}$$

$$\ln a_{l,k+1} = \ln a_{l,k} - \ln 2 + 2\ln z - \ln(k+1) - \ln(2(l+k)+3) \tag{10.85}$$

The procedure is to compute $a_{l,0}$ (up to a given L) using (10.84), then recurse over k using (10.85) until convergence. The convergence will be different for each l. Using $k = 0$ in (10.84), $a_{l,0}$ are computed with:

$$\ln a_{l+1,0} = \ln a_{l,0} + \ln z - \ln(2l+3) \tag{10.86}$$

where $\ln a_{0,0} = -z$.

Routine The routine `gaussianRandomParticle` takes as input the coordinates (θ, ϕ), mean radius of the particle, log-normal surface RMS, and correlation angle, and returns the radius of the surface at each spherical point computed with the methods above. The maximum degree L is optional and defaults to the heuristic. It computes the harmonic sum directly to allow input of any sampling of (θ, ϕ), but this is inefficient for a large number of points and small correlation angles (because L is large), even though the computation is correct. If the spherical angles can be sampled at the points of quadrature, the particle surface could be computed using a fast spherical transform, which is mentioned here for future development. Alternatively, a version of this in which the Legendre polynomials are computed on the fly with inline recursion could be developed to save memory.

```
function [R] = gaussianRandomParticle(Theta,Phi,a,rmsh,Gamma_deg,L)
% Compute surface radii of a Gaussian random particle
%
% Theta,Phi:    [Any size] Spherical angles (radians) at which to compute the surface
% a:            Mean radius of surface (length)
% rmsh:         log-normal RMS of the surface (length)
% Gamma_deg:    Correlation angle (degrees)
% L:            [optional] Maximum harmonic degree L, defaults to heuristic
%
% R:            [Same size as Theta,Phi] Surface radius at (Thtea,Phi)
%
% Dependencies: coefclp, prodexpil

sigma = rmsh/a;                 % sigma parameter
beta = sqrt(log(sigma^2 + 1));  % beta parameter (std of zero mean Gaussian)
if nargin == 5
    L = round(275/Gamma_deg + 2.5); % number of harmonics given Gamma from heuristic
end

% harmonic indexing
tot = sum(0:L) + L + 1;         % total number of harmonics
lm = zeros(tot,2);              % store (l,m) pairs in lm
cnt = 0;
for l=0:L,
for m=0:l,
    cnt = cnt+1;
    lm(cnt,1) = l;
    lm(cnt,2) = m;
end
end

% pre-compute normalized associated Legendre polynomials
Leg = [];
for l=0:L,
    tmp = legendre(l,cos(Theta(:)),'norm');
    Leg = [Leg; tmp];
end

sz = size(Theta);
Theta = Theta(:)';
Phi = Phi(:)';
% make m index and phi gridded
[M P2] = ndgrid(0:L,Phi);
% pre-compute cosine and sine functions
C = cos(M.*P2);
S = sin(M.*P2);

% random draws for alm' and blm' from standard normal
% this indexing ensures that the seed counts up from the lowest harmonic
tmp = randn(2*tot,1);
almp = tmp(1:2:end);
blmp = tmp(2:2:end);

% compute cl'
[clp] = coefclp(L,Gamma_deg);
```

```
% sum up the harmonics with correct coefficients
s = zeros(size(Theta));
ind = 0;
for l=0:L,
for m=0:l,
    ind = ind+1;
    if m==0
        const = sqrt(2);
    else
        const = 1;
    end
    tmp = clp(l+1)/const*(almp(ind)*C(m+1,:) + blmp(ind)*S(m+1,:));
    s = s + Leg(ind,:).*tmp;
end
end
s = beta*s;

% log-normal radius
R = a/sqrt(1 + sigma^2)*exp(s);
R = reshape(R,sz);

end

function [clp] = coefclp(L,Gamma)
% compute expansion coefficients of the correlation function
    lc = 2*sind(Gamma/2);
    x = 1/lc^2;
    tmp = prodexpil(L,x);
    clp = sqrt(2)*sqrt(2.*tmp);
end

function [S] = prodexpil(L,z)
% compute the product of exp(-z)*i_l(z)
% by log transform with recursion
%
% S = exp(-z)*sqrt(pi/(2*z))*besseli((0:L)'+1/2,z)

lnz = log(z);
lnal0 = zeros(L+1,1);
% iterate a_{l+1,0)} inital conditions
lnal0(1) = -z;
for l = 0:(L-1),
   ind = l+1;
   lnal0(ind+1) = lnal0(ind) + lnz - log(2*l+3);
end

% iterate the sums
S = zeros(L+1,1);
for l = 0:L,
    v = lnal0(l+1);
    g = exp(v);
    first = v;
    k = 0;
    cnvg = 0;
    const = -log(2) + 2*lnz;
    while cnvg == 0
        v = v + const - log(k+1) - log(2*(l+k)+3);
        g = g + exp(v);
        k = k + 1;
        if v < (first - 10)
            cnvg = 1;
        end
    end
    S(l+1) = g;
end
end
```

10.8 Ionosphere Irregularity

Background Earth's ionosphere is a complex plasma medium that exists at altitudes between 100 km and 600 km and affects the propagation of radio, radar, and GPS signals. The influence of the ionosphere on these signals increases as the radio frequency decreases. It starts to affect radar around L-band (1-2 GHz) and UHF-band (300 MHz - 1 GHz), becomes a significant dispersive medium in the VHF band (30-300 MHz), and is completely reflecting in the HF band below the plasma frequency (around 10 MHz). The strength of the ionosphere is highly dependent on the time of day and latitude. The ionosphere is always changing, but a given 3D spatial distribution of electron density can usually be considered constant over a short time duration, for example, of a synthetic aperture radar acquisition from an orbiting spacecraft.

The simplest parameters that quantify the ionosphere are 1) electron density, N_e, which is a function of altitude and given in terms of number of electrons per unit volume (el/m^3), 2) the total electron content (TEC) which is the column-integrated electron density up to a given altitude, given in terms of number of elections per unit area (el/m^2), and 3) irregularities that are random fluctuations in the spatial distribution of the plasma density. The TEC is given by

$$TEC(h) = \int_0^h N_e(z)dz \tag{10.87}$$

where $N_e(z)$ is the electron density as a function altitude, and h is the altitude up to which we wish to evaluate the TEC. TEC is often the only parameter needed to assess the two-way effects of the ionosphere on the radar signal such as dispersion, Faraday rotation, and absorption loss. Irregularities contribute to scintillation, which are fluctuations of the phase and amplitude of the radar echo about a mean response. Scintillation creates small phase variations across a synthetic aperture, and these phase variations lead to decoherence and a drop in coherent gain when creating images. While strong ionospheric dispersion at VHF frequencies is almost completely correctable, scintillation is often uncorrectable, because there is no way to predict what the irregularities will be at a given time or place. Finally, the study of Earth's ionosphere spans many decades, and more information on these topics can be found in the literature.

Formulation We give a routine for creating a 1D profile of ionospheric irregularities that is suitable for radar analysis. This is based on the 2-parameter PSD model in [72, 73], from which 1D profiles can be created using the methods from Section 10.1. The 2-parameter model effectively creates different regimes of roughness that define the underlying structure of the irregularity. These are controlled through two log-slope regions in the PSD. The log-slope values come from empirical observations, while the mean and relative variation are set by user-defined scale factors. The signal can be scaled to either electron density, assuming a uniform column, or to the TEC itself. The scale factors in this model are not very informative, however, computing actual TEC and irregularity amplitude is quite complicated. A more informative model is given, for example, in [74], which provides a PSD of radar phase scintillation using scale factors from up-to-date global ionospheric models. Still, the use of user-defined scale factors can suffice for first-order analysis.

From [72, 73], the 2-parameter PSD for the electron density irregularity is given for 3D k-space as

$$\Phi(k) = \begin{cases} \dfrac{A}{(k^2 + k_o^2)^{v_1}} & k \leq k_b \\ \dfrac{A(k_o^2 + k_b^2)^{v_2 - v_1}}{(k^2 + k_o^2)^{v_2}} & k > k_b \end{cases} \tag{10.88}$$

where k is the spatial wavenumber, $k_o = 2\pi/L_o$ is the outerscale wavenumber at length scale L_o, $k_b = 2\pi/L_b$ is the break wavenumber at length scale L_b, v_1 and v_2 are log-slope parameters, and A is a scale factor. The values for the parameters are empirical and cited in [73] as: $L_o \approx 10$ km, $L_b \approx 500$ m, $2v_1 \approx$ 3-3.5, and $2v_2 \approx$ 5-5.5.

The 1D PSD, $V(k)$, is related to the 3D PSD as [75]

$$\Phi(k) = -\frac{1}{2\pi k}\frac{\partial V(k)}{\partial k} \tag{10.89}$$

which leads to the 1D PSD, [72],

$$V(k_x) = \begin{cases} \dfrac{\pi A}{v_1 - 1} \dfrac{1}{(k_x^2 + k_o^2)^{v_1 - 1}} - \dfrac{\pi A}{(k_b^2 + k_o^2)^{v_1 - 1}} \left(\dfrac{1}{v_1 - 1} - \dfrac{1}{v_2 - 1} \right) & k_x \leq k_b \\ \dfrac{\pi A (k_o^2 + k_b^2)^{v_2 - v_1}}{v_2 - 1} \dfrac{1}{(k_x^2 + k_o^2)^{v_2 - 1}} & k_x > k_b \end{cases} \tag{10.90}$$

where k_x is now a 1D wavenumber. The key feature of this PSD are two regimes of log-slope power.

The 1D profile of electron density, $N_e(x)$, is realized by multiplying each frequency of the PSD with a draw from a complex standard normal distribution and then taking the inverse Fourier transform. The amplitude, A, has so far been arbitrary, but it is there to adjust the scale of the signal. Let $n_e(x)$ be a zero mean profile generated by (10.90) which is normalized by its RMS. This signal can then be rescaled as

$$N_e(x) = \overline{N}_e \left(\tilde{\sigma} n_e(x) + 1 \right) \tag{10.91}$$

where $\overline{N}_e$ is the average electron density and $\tilde{\sigma}$ is the relative RMS variation given as a fraction, or percentage, of the mean. This assumes that the ionospheric column has uniform electron density. Alternatively, (10.91) can be used for TEC if $\overline{N}_e$ and $\tilde{\sigma}$ are replaced by the average TEC and percentage RMS variation in TEC, respectively.

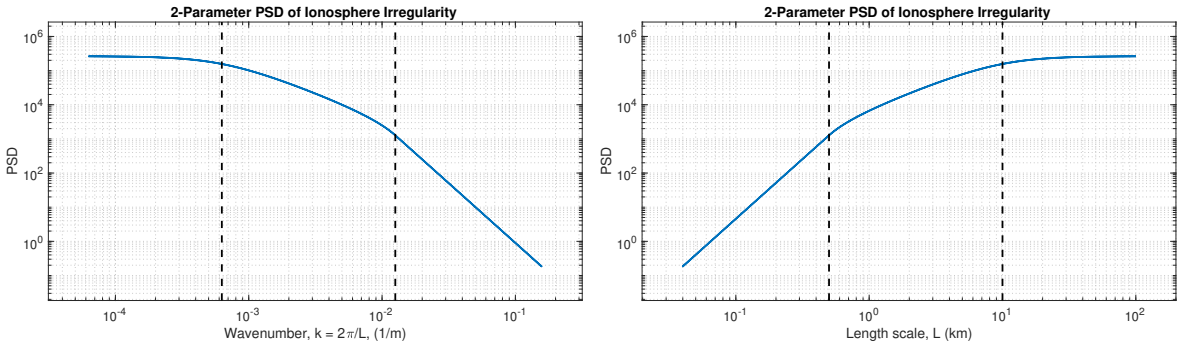


Figure 10.11: PSD of ionosphere irregularity with respect to wavenumber (left) and length scale (right) for the default parameters: $L_o = 10$ km, $L_b = 500$ m, $2v_1 = 3.5$, and $2v_2 = 5.5$. Dashed lines are the outerscale and breakscale. The end points of the curves correspond to the wavenumber or length scale of the total profile length or the sampling step.

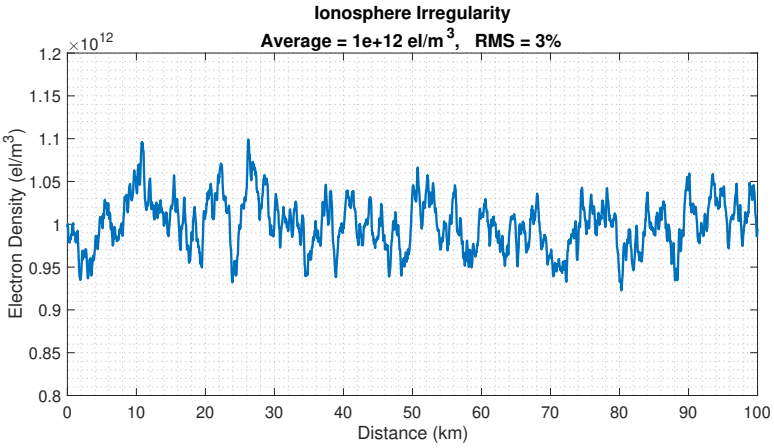


Figure 10.12: Profile of ionosphere irregularity for the default parameters: $L_o = 10$ km, $L_b = 500$ m, $2v_1 = 3.5$, and $2v_2 = 5.5$. Numerically, the correlation length was found to be about 1 km.

Routine The routine `ionosphere1` produces a 1D profile of ionosphere irregularity based on the 2-parameter PSD above. It takes as input the number of sample points, profile length, average value, and relative RMS variation in percentage of the mean. The scale factors can be electron density or TEC. It uses the same generating procedure as `rough1` in Section 10.1. Default model parameters are: $L_o = 10$ km, $L_b = 500$ m, $2v_1$ =3.5, and $2v_2$ =5.5. Different values can be optionally included as input arguments, in the same order, where an input of `'[]'` will use the default value. The routine can output the PSD, wavenumber, and model parameters for plotting.

```
function [Ne V kx Lo Lb v1 v2] = ionosphere1(Nx,Lx,ave,pctrms,varargin)
% 1D profile of ionosphere irregularity using 2-parameter model
%
% N:            N number of points
% L:            Profile length (m)
% ave:          Average value of electron density or TEC
% pctrms:       Percent (%) RMS of electron density or TEC deviation
% varargin:     [optional] Lo, Lb, v1, v2
%               Lo, Lb: Outerscale and breakscale (m)
%               v1, v2: log-slope parameters
%
% Ne:           Nx1 array of ionosphere irregularity profile (electron density, Ne, or TEC)
% V,kx:         [optional] Spectral density and wavenumber
% Lo,Lb,v1,v2:  [optional] will output default values if needed
%
% Dependencies: fftfreq

dx = Lx/(Nx-1);
kx = 2*pi*fftfreq(1/dx,Nx);

% spectral length scales and log-slope parameters
% defaults
Lo = 10e3;      % outerscale, 10 km
Lb = 500;       % breakscale, 500 m
v1 = 3.5/2;     % log-slope parameter 1
v2 = 5.5/2;     % log-slope parameter 2

% user defined parameters
if length(varargin)>=1 && ~isempty(varargin{1}), Lo = varargin{1}; end
if length(varargin)>=2 && ~isempty(varargin{2}), Lb = varargin{2}; end
if length(varargin)>=3 && ~isempty(varargin{3}), v1 = varargin{3}; end
if length(varargin)==4 && ~isempty(varargin{4}), v2 = varargin{4}; end

% wavenumbers
ko = 2*pi/Lo;   % outerscale wavenumber
kb = 2*pi/Lb;   % breakscale wavenumber

% initialize spectral density, V
V = zeros(Nx,1);

% compute V at lower wavenumbers
ind = (abs(kx) <= kb);
V(ind) = (pi/(v1-1))./((kx(ind).^2+ko^2).^(v1-1)) ...
    - pi/((kb^2+ko^2)^(v1-1))*(1/(v1-1)-1/(v2-1));

% compute V at upper wavenumbers
ind = (abs(kx) > kb);
V(ind) = (pi/(v2-1))*((ko.^2+kb^2)^(v2-v1))./((kx(ind).^2+ko^2).^(v2-1));

% create unnormalized profile
gam = randn(Nx,1) + 1i*randn(Nx,1);
gam(1) = 0;
H = gam.*sqrt(V);
dkx = 2*pi/Lx;
Ne = (Nx*sqrt(dkx))*real(ifft(H));

% normalize by RMS and apply scale factors
Ne = Ne/rms(Ne);
Ne = ave*(pctrms/100*Ne + 1);
```

Chapter 11

Reflection and Refraction

This chapter contains routines for geometric optics solutions of reflection and refraction of rays at dielectric interfaces. All of these solutions are fundamentally based on Snell's law. Specifically, these routines will compute the reflection or reflection point on an interface given the positions of a source and observer as inputs. These algorithms are especially useful for radar scattering and subsurface imaging problems. Many of these solutions are well known. In general, very fast, and much better, codes for geometric optics can be found in the computer graphics literature.

This chapter contains: 1) routines for computing the reflection point on a plane and sphere, 2) derivations and two routines for solving for the refraction point through the plane: one is analytical, one is iterative, 3) an analytical solution for the refraction point on a circular interface given a pair of interior and exterior source and observation points, 4) a vectorized iterative solution for refraction through a circular interface, and 5) a simple coordinate transform for the solution for the refraction point on a sphere.

11.1 Snell's Law

Snell's law of refraction at a flat dielectric half-space is given by

$$\frac{\sin\theta_1}{\sin\theta_2} = \frac{n_2}{n_1} \tag{11.1}$$

where θ_1 is the angle measured from normal in medium 1 with index of refraction, n_1, and θ_2 is the angle measured from normal in medium 2 with index of refraction, n_2. Recall $n = \sqrt{\epsilon_r}$, where ϵ_r is the relative permittivity.

11.2 Reflection Point - Flat Interface

Here we find the ray-solution for the specular point for two arbitrary points above a flat surface. One point could be a source, the other a receiver. This can be solved a number of ways, including the image method or by enforcing equal incident and reflected angles relative to the normal of the specular point.

Let two points $\mathbf{r}_1$ and $\mathbf{r}_2$ be above a flat surface that is parallel to the XY plane at level $z = z_o$. Each half space has a different index of refraction. Define the following quantities

$$h_1 = z_1 - z_o \tag{11.2}$$

$$h_2 = z_2 - z_o \tag{11.3}$$

$$L = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \tag{11.4}$$

$$\hat{u} = \frac{(x_2 - x_1)\hat{x} + (y_2 - y_1)\hat{y}}{L} \tag{11.5}$$

where $\hat{u}$ is the transverse unit vector from $\mathbf{r}_1$ and $\mathbf{r}_2$. Define the transverse distance from $\mathbf{r}_1$ to the reflection point as u_1 and from $\mathbf{r}_2$ to the reflection point as u_2. Then for equal incident and reflected angles relative

to normal

$$\tan\theta = \frac{u_1}{h_1} \tag{11.6}$$

$$\tan\theta = \frac{u_2}{h_2} \tag{11.7}$$

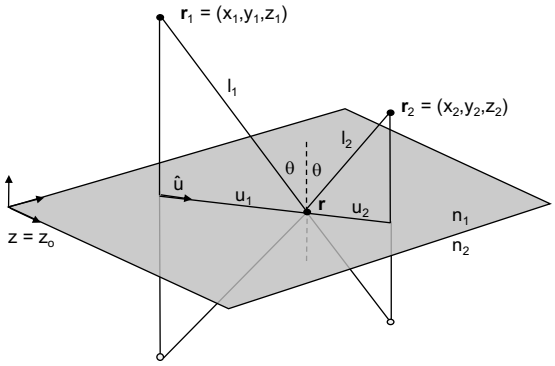


Figure 11.1: Geometry for the solution of the planar reflection point, $\mathbf{r}$, which is unknown.

Equating the sines, using the fact that $L = u_1 + u_2$, and solving for u_1, we get

$$u_1 = \frac{h_1 L}{(h_2 + h_1)} \tag{11.8}$$

From which the reflection point $\mathbf{r}$ is found by reprojecting

$$\mathbf{r} = x_1\hat{x} + y_1\hat{y} + u_1\hat{u} + z_o\hat{z_o} \tag{11.9}$$

The routine `reflectionPlane` takes as input the two points, $\mathbf{r}_1$, $\mathbf{r}_2$, and the level of the plane, z_o, and returns the reflection point, $\mathbf{r}$, as well as θ and the lengths of the vectors l_1 and l_2. This routine is vectorized for any number of pairs of source and receiver points.

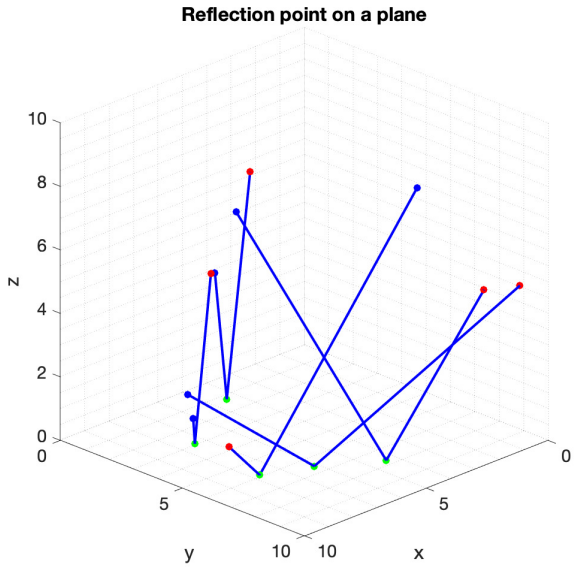


Figure 11.2: Geometry for the solution of the planar reflection point, $\mathbf{r}$, which is unknown off a plane at level $z_o = 0$.

```
function [rx ry rz theta l1 l2] = reflectionPlane(x1,y1,z1,x2,y2,z2,zo);
% Reflection point on a plane
%
% x1,y1,z1      Coordinates of exterior point, r1, centered on circle
% x2,y2,z1      Coordinates of interior point, r2, centered on circle
% zo            z-level of the plane
%
% rx,ry,rz      Coordinates of refraction point
% theta         Reflection angle
% l1,l2         Lengths of the vectors from each point to the reflection point

% note bad z-coordinates
ind = or((z1 <= 0),(z2 <= 0));

% compute parameters
h1 = z1 - zo;
h2 = z2 - zo;
L = sqrt((x2-x1).^2 + (y2-y1).^2);
uhat_x = (x2-x1)./L;
uhat_y = (y2-y1)./L;

% zero out points on top of each other
indL = (L==0);
uhat_x(indL) = 0;
uhat_y(indL) = 0;

% solution to u1
u1 = h1.*L./(h2 + h1);
u2 = L - u1;

% compute reflection point
rx = x1 + u1.*uhat_x;
ry = y1 + u1.*uhat_y;
rz = zo*ones(size(x1));

% compute other quantities
theta = atan2(u1,h1);
l1 = sqrt(u1.^2 + h1.^2);
l2 = sqrt(u2.^2 + h2.^2);

% nan-out bad z-coordinates
rx(ind) = nan;
ry(ind) = nan;
rz(ind) = nan;
theta(ind) = nan;
l1(ind) = nan;
l2(ind) = nan;
```

11.3 Reflection Point - Sphere

Computing the reflection point on a sphere is often needed in Earth and planetary remote sensing, for example, when determining the specular point on a body between a source and receiver. The solution here is the one derived in [76], and summarized next. The reflection point on a circle can be found with this method by restricting the source and receiver points to a plane.

Let a sphere with radius r be centered at the origin. $\mathbf{r}_1$ and $\mathbf{r}_2$ are radius-normalized vectors to exterior points 1 and 2, respectively, which can be a source and receiver. Next, define the following coefficients

$$a = \mathbf{r}_1 \cdot \mathbf{r}_1 \tag{11.10}$$

$$b = \mathbf{r}_1 \cdot \mathbf{r}_2 \tag{11.11}$$

$$c = \mathbf{r}_2 \cdot \mathbf{r}_2 \tag{11.12}$$

After enforcing the reflection condition on the sphere, the solution comes from finding the roots of the

following quartic polynomial

$$\sum_{n=0}^{4} a_n y^n = 0 \tag{11.13}$$

$$\begin{aligned} a_4 &= 4c(ac - b^2) && (11.14)\\ a_3 &= -4(ac - b^2) && (11.15)\\ a_2 &= a + 2b + c - 4ac && (11.16)\\ a_1 &= 2(a - b) && (11.17)\\ a_0 &= a - 1 && (11.18) \end{aligned}$$

Once the roots are known, the positive roots with zero imaginary part are kept, call these $\bar{y}$. This is used to compute a second quantity

$$\bar{x} = \frac{-2c\bar{y}^2 + \bar{y} + 1}{2b\bar{y} + 1} \tag{11.19}$$

The solution is the pair $(\bar{x}, \bar{y})$ for which both are positive. Then the reflection point on the sphere is

$$\mathbf{r} = r(\bar{x}\mathbf{r}_1 + \bar{y}\mathbf{r}_2) \tag{11.20}$$

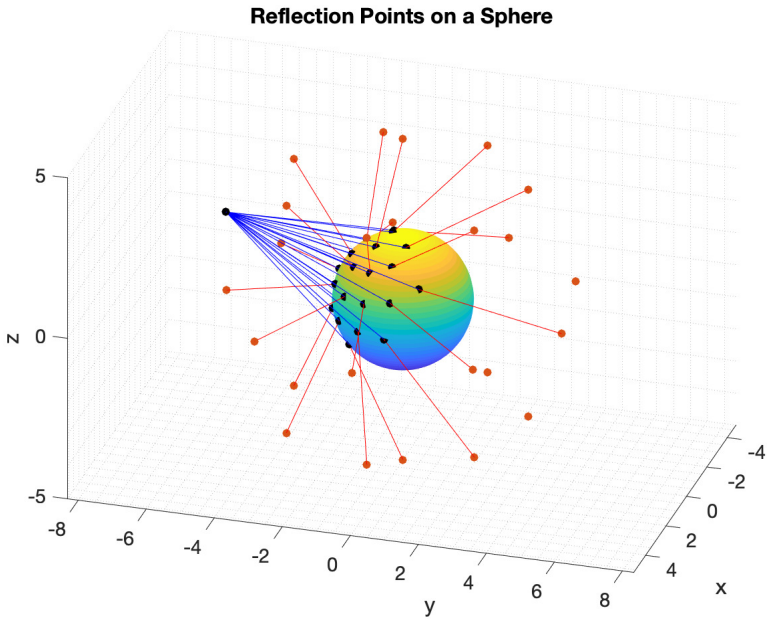


Figure 11.3: Geometry for the reflection point on a sphere.

Line-Sphere Intersection We exclude shadowed points by using the line-sphere intersection solution, [77]. The equation for a line that originates at $\mathbf{r}_1$ and passes through $\mathbf{r}_2$ is

$$\mathbf{u}(l) = \mathbf{r}_1 + l\hat{u} \tag{11.21}$$

where l is the distance along the line and $\hat{u}$ is

$$\hat{u} = \frac{\mathbf{r}_2 - \mathbf{r}_1}{|\mathbf{r}_2 - \mathbf{r}_1|} \tag{11.22}$$

For a sphere with center, $\mathbf{c}$, and radius, r, the solutions for l are

$$l = -\hat{u} \cdot (\mathbf{r}_1 - \mathbf{c}) \pm \sqrt{\Delta} \quad (11.23)$$

$$\Delta = (\hat{u} \cdot (\mathbf{r}_1 - \mathbf{c}))^2 - \left(\|\mathbf{r}_1 - \mathbf{c}\|^2 - r^2\right) \quad (11.24)$$

If $\Delta < 0$ then there are no intersections. If $\Delta = 0$ then the line is tangent to the sphere. If $\Delta > 0$ then the line intersects the sphere at two places. In the last case, the reflection point is determined by checking which of the two intersection points is closer to the source.

Routine The routine `reflectionSphere` solves for the reflection point on a sphere that is centered at the origin. It takes as input the Cartesian points $\mathbf{r}_1$, $\mathbf{r}_2$, and the sphere radius, r, and returns the reflection point, $\mathbf{r}$, which lies on the surface of the sphere. The routine is vectorized to take multiple pairs of points at once, and returns `nan` if the points are shadowed or interior to the sphere. Shadowing is determined by the routine `lineIntersectSphere`, not copied here, that implements the line-sphere intersection solution above.

```
function [rx ry rz] = reflectionSphere(x1,y1,z1,x2,y2,z2,r)
% Reflection point on a sphere centered at the origin
%
% x1,y1,z1  Cartesian positions of first point (any size)
% x2,y2,z2  Cartesian positions of second point (same size as x1,y1,z1)
% r:        Sphere radius
%
% rx,ry,rz: Vector to the reflection point on sphere
%
% Based on David Eberly, Geometric Tools, 2008
%
% Dependencies: lineIntersectSphere,quarticroots

% check that points are outside sphere
sz = size(x1);
r1 = [x1(:) y1(:) z1(:)];
r2 = [x2(:) y2(:) z2(:)];
N = length(x1(:));

% points that are internal to the sphere
isbad = or((sqrt(sum(r1.^2,2)) <= r),(sqrt(sum(r2.^2,2)) <= r));

% points for which the line intersects the sphere (therefore shadowed)
[~, ~, ~, ~, delta_ind] = lineIntersectSphere(r1,r2,r);
isintersect = zeros(N,1);
isintersect(delta_ind == 1) = 1;

% check to see if those points are in front or behind
isshaddow = and((sign(dot(r1,r2,2)) == -1),isintersect);

% normalize points to radius of sphere
L = r1/r;
S = r2/r;
a = dot(S,S,2);
b = dot(S,L,2);
c = dot(L,L,2);
% fill coefficient array
qa = 4*c.*(a.*c-b.^2);
qb = -4*(a.*c-b.^2);
qc = a+2*b+c-4*a.*c;
qd = 2*(a-b);
qe = a-1;
[x1 x2 x3 x4] = quarticroots(qa,qb,qc,qd,qe);
ybar = [x1 x2 x3 x4];

% solve for projection scale factors
xbar = (-2*c.*ybar.^2 + ybar + 1)./(2*b.*ybar+1);
u = zeros(N,2);
for n=1:4,
    ind = and(and(imag(ybar(:,n))==0,ybar(:,n)>0),xbar(:,n)>0);
```

```
    u(ind,1) = xbar(ind,n);
    u(ind,2) = ybar(ind,n);
end
xbar = u(:,1);
ybar = u(:,2);

% reproject
rr = r*(xbar.*S + ybar.*L);
rr(or(isbad,isshaddow),:) = nan;
rx = reshape(rr(:,1),sz);
ry = reshape(rr(:,2),sz);
rz = reshape(rr(:,3),sz);
```

11.4 Refraction Point - Flat Interface

We derive two solutions for the refraction point through a flat dielectric interface given two arbitrary points on either side of the interface. The first solution is in terms of the roots of a quartic polynomial, [78]. The second is an iterative algorithm, [79]. Practical applications of this are subsurface SAR processing, e.g. radar sounding through ice, ground penetrating radar, or through-wall imaging, where the propagation phase between two points across a dielectric interface needs to be computed for focusing. In these cases, the refraction point is really an intermediate result in order to obtain the incidence and transmission angles and propagation distances along the bent-ray path.

The geometry is shown in Figure 11.4. The dielectric interface is parallel to the XY plane at level $z = z_o$. Let two points $\mathbf{r}_1$ and $\mathbf{r}_2$ be above and below the interface in mediums 1 and 2 each having an index of refraction, n_1 and n_2, respectively. Next define the following quantities,

$$h = z_1 - z_o \tag{11.25}$$

$$d = z_2 - z_o \tag{11.26}$$

$$L = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \tag{11.27}$$

$$\hat{u} = \frac{(x_2 - x_1)\hat{x} + (y_2 - y_1)\hat{y}}{L} \tag{11.28}$$

where h is the height of $\mathbf{r}_1$ above the interface, d is the depth of $\mathbf{r}_2$ below the interface, $\hat{u}$ is the transverse unit vector and L is the transverse distance from point 1 to point 2. Finally, u is the transverse distance along $\hat{u}$ between the projection of $\mathbf{r}_1$ on the interface and the refraction point.

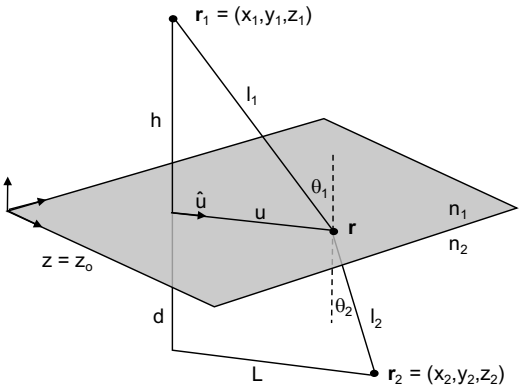


Figure 11.4: Geometry for the solution of the planar refraction point, $\mathbf{r}$, which is unknown.

Quartic Solution

Start with the observations that

$$\begin{aligned} \sin\theta_1 &= \frac{u}{\sqrt{h^2+u^2}} \\ \sin\theta_2 &= \frac{L-u}{\sqrt{d^2+(L-u)^2}} \end{aligned}$$

These are equated though Snell's law

$$\frac{u}{\sqrt{h^2+u^2}} = \frac{n_2}{n_1}\frac{L-u}{\sqrt{d^2+(L-u)^2}} \tag{11.29}$$

After squaring both sides, this can be rearranged as a fourth order polynomial in u:

$$au^4 + bu^3 + cu^2 + du + e = 0 \tag{11.30}$$

$$\begin{aligned} a &= n_1^2 - n_2^2 \\ b &= -2L(n_1^2 - n_2^2) \\ c &= n_1^2(L^2+d^2) - n_2^2(L^2+h^2) \\ d &= 2n_2^2 L h^2 \\ e &= -n_2^2 L^2 h^2 \end{aligned}$$

The roots can be found with a root finding algorithm or analytically. The solution is the positive real root for which $u < L$. This solution is good for either $n_1 > n_2$ or $n_1 < n_2$. Once found, u is reprojected to give the refraction point:

$$\mathbf{r} = x_1\hat{x} + y_1\hat{y} + u\hat{u} + z_o\hat{z} \tag{11.31}$$

Iterative Solution

For the iterative solution, we use the facts that

$$u = h\tan\theta_1 \tag{11.32}$$

$$u = L - d\tan\theta_2 \tag{11.33}$$

Start with the straight-ray approximation for θ_1, then with the addition of Snell's law, we can build the following iteration

$$\begin{aligned} \text{Initialize:} \quad \theta_1 &\leftarrow \arctan\left(\frac{L}{h+d}\right) \\ \text{Iterate:} \quad \theta_2 &\leftarrow \arcsin\left(\frac{n_1}{n_2}\sin\theta_1\right) \\ u &\leftarrow L - d\tan\theta_2 \\ \theta_1 &\leftarrow \arctan\left(\frac{u}{h}\right) \end{aligned}$$

This is an alternating sequence for u (and θ_1, θ_2). The physical interpretation is that the value of u bounces closer to the true solution at every iteration. This is effectively a Newton iteration of the transcendental equation for θ_1 without a nonlinear optimization algorithm. The solution is lightning fast (requiring maybe 10 iterations), when it works. The solution does not always converge, which happens when h is too close to the interface relative to values of d and L. In other words, this solution is valid when $d, L \leq h$. Also, as written, this should only be used when $n_1 < n_2$.

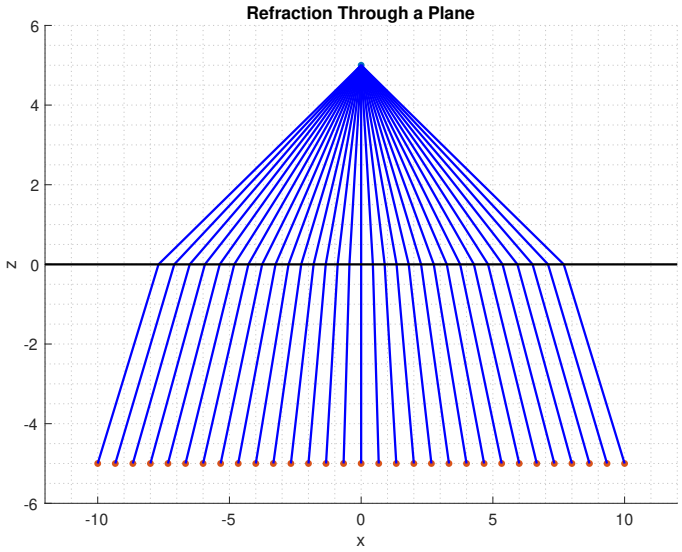


Figure 11.5: Refraction through a plane, $n_1 = 1$, $n_2 = 2$.

Routine

The routine `refractionPlane` takes as input the coordinates of $\mathbf{r}_1$, $\mathbf{r}_2$, z_o, n_1 and n_2, and returns the coordinates of the refraction point $\mathbf{r}$ on the interface, the angles θ_1, θ_2, and lengths of the vectors to the refraction point, l_1 and l_2. The quartic solution is used by default. Use the optional string switch `'it'` for the iterative solution. For the latter, the routine estimates the required number of iterations by testing a point with the largest free-space θ_1, then applies that number of iterations to all other points in parallel. The routine is vectorized to take one point, $\mathbf{r}_1$, while the coordinates of subsurface points, $\mathbf{r}_2$, can be arrays of any size (all the same size). Outputs are the same size as the $\mathbf{r}_2$ coordinates. This assumes n_1 and n_2 are real. The case when $\mathbf{r}_2$ is at or above the boundary is handled separately.

```
function [RX RY RZ theta1 theta2 l1 l2] = refractionPlane(x1,y1,z1,X2,Y2,Z2,zo,n1,n2,str);
% Refraction through a dieletric interface
% Interface is parallel to XY plane. Assumes n1, n2 are real.
% Default solution is via quartic, with optional iterative solution
%
% x1,y1,z1  1x1 Cartesian coordinates of point in medium 1
% X2,Y2,Z2  [any size] Cartesian coordinates of points in medium 2
% zo        z coordinate of dielectric interface
% n1,n2     Index of refraction in medium 1 and 2
% str       [optional] 'it' for iterative solution
%
% RX,RY,RZ  [size(X2)] Cartesian coordinates of refraction point
% theta1    [size(X2)] Refraction angle in medium 1 (radians)
% theta2    [size(X2)] Refraction angle in medium 2 (radians)
% l1        [size(X2)] Distance from r1 to r
% l2        [size(X2)] Distance from r2 to r
%
% Dependencies: iterateRefractionTest, iterateRefraction

sz = size(X2);
N = numel(X2);
n1 = real(n1);
n2 = real(n2);

if n1 < 1 || n2 < 1
    disp('Error: bad index of refraction')
    return
end
if z1 <= zo
    disp('Error: bad z coordinate')
```

```
    return
end

% projection onto vertical plane
x21 = X2 - x1;
y21 = Y2 - y1;
L = sqrt(x21.^2 + y21.^2);
uhat_x = x21./L;
uhat_y = y21./L;
H = abs(z1 - zo);
D = abs(Z2 - zo);

if nargin == 9              % quartic solution
    % quartic coefficients
    a = (n1.^2 - n2.^2)*ones(sz);
    b = -2*L.*a;
    c = n1.^2.*(L.^2 + D.^2) - n2.^2.*(L.^2 + H.^2);
    d = 2*n2.^2.*L.*H.^2;
    e = -n2.^2.*L.^2.*H.^2;
    % quartic roots vectorized
    [w1 w2 w3 w4] = quarticroots(a,b,c,d,e);
    % pick out the one real, positive, valid root
    r = [w1(:) w2(:) w3(:) w4(:)];
    u = zeros(N,1);
    for n=1:4,
        ind = and(and(imag(r(:,n))==0,r(:,n)>0),r(:,n)<L(:));
        u(ind) = r(ind,n);
    end
    u = reshape(u,sz);
elseif nargin == 10 && strcmp(str,'it')   % iterative solution
    % maximum straight ray theta1
    [M I] = max(atan2(L,abs(Z2-z1)));
    % test for number of iterations
    nit = iterateRefractionTest(L(I),H,D(I),n1,n2);
    % use nit iterations for all points
    u = iterateRefraction(L,H,D,n1,n2,nit);
end

% reproject the refraction point in original frame
RX = x1 + uhat_x.*u;
RY = y1 + uhat_y.*u;
RZ = zo*ones(sz);

% compute angles and lengths
theta1 = real(asin(u./sqrt(H.^2 + u.^2)));
theta2 = real(asin((L - u)./(D.^2 + (L - u).^2)));
l1 = sqrt(u.^2 + H.^2);
l2 = sqrt(D.^2 + (L - u).^2);

% if subsurface points are vertically algined with upper point
% expect this number to be small so use find
% other parameters are taken care of by L
ind = find(L==0);
RX(ind) = x1;
RY(ind) = y1;
theta1(ind) = 0;
theta2(ind) = 0;
l1(ind) = H;
l2(ind) = D(ind);

% take care of points above or equal to boundary
ind = find(Z2 >= zo);
RX(ind) = nan;
RY(ind) = nan;
RZ(ind) = nan;
theta1(ind) = real(asin(L(ind)./sqrt((H-Z2(ind)).^2 + L(ind).^2)));
theta2(ind) = nan;
l1(ind) = sqrt(L(ind).^2 + (H-Z2(ind)).^2);
```

```
l2(ind) = nan;

% take care of points on boundary
ind = find(Z2 == zo);
RX(ind) = X2(ind);
RY(ind) = Y2(ind);
RZ(ind) = Z2(ind);
end


function [it] = iterateRefractionTest(L,h,d,n1,n2);
% Testing function that returns the number of iterations for one point
    th1 = atan2(L,h+d);
    it = 0;
    ratio = 1;
    while ratio > 1e-8
        th2 = asin((n1/n2)*sin(th1));
        u = L-d.*tan(th2);
        th1new = atan2(u,h);
        ratio = abs(th1new-th1);
        th1 = th1new;
        it = it+1;
    end
end

function [u] = iterateRefraction(L,h,d,n1,n2,nit);
% Vectorized iteration solution for given number of iterations, nit
    th1 = atan2(L,h+d);
    u = d.*tan(th1);
    for n=1:nit
        th2 = asin((n1/n2)*sin(th1));
        u = L-d.*tan(th2);
        th1 = atan2(u,h);
    end
end
```

11.5 Refraction Point - Circular Interface

We provide two solutions for the refraction point on circular interface. The first is a semi-analytical solution in terms of the roots of a sixth degree polynomial. The second is a vectorized iterative solution that is valid for weak refraction.

11.5.1 Refraction Point - Circular Interface - Analytical

A semi-analytic solution for the refraction point on a circular interface is derived for an arbitrary interior and exterior point. This is based on satisfying Snell's law at the interface. Similar to the quartic solution for the refraction point on a flat surface, the solution is found in terms of the roots of a sixth degree polynomial. An alternative approach is to apply the principle of least action to the total electrical path length of the rays. Both approaches give the same result.

The geometry is shown in Figure 11.6. Let a circle be centered at the origin with radius r. The indices of refraction outside and inside the circle are n_1 and n_2, respectively. Let the exterior and interior points be, $\mathbf{r}_1$, $\mathbf{r}_2$, and the refraction point on the circle, $\mathbf{r}$. Arbitrary exterior/interior points are rotated so that the exterior point is aligned with the y axis. Once the refraction point is found in this geometry, it is rotated back. Start with

$$\mathbf{r}_1 = y_1\hat{y} \tag{11.34}$$

$$\mathbf{r}_2 = x_2\hat{x} + y_2\hat{y} \tag{11.35}$$

$$\mathbf{r} = r_x\hat{x} + r_y\hat{y} \tag{11.36}$$

where $y_1 > r$ and $r^2 = r_x^2 + r_y^2$. Also, define the vectors $\mathbf{v}_1 = \mathbf{r}_1 - \mathbf{r}$ and $\mathbf{v}_2 = \mathbf{r}_2 - \mathbf{r}$. The sine of the incident

and transmission angles at the refraction point can be written as the following cross products:

$$\sin\theta_1\hat{z} = \frac{\mathbf{r}\times\mathbf{v}_1}{|\mathbf{r}||\mathbf{v}_1|} \tag{11.37}$$

$$\sin\theta_2\hat{z} = \frac{(-\mathbf{r})\times\mathbf{v}_2}{|\mathbf{r}||\mathbf{v}_2|} \tag{11.38}$$

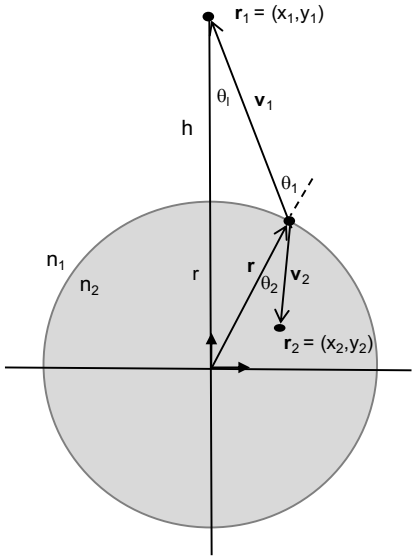


Figure 11.6: Geometry for refraction through a circle.

Equating through Snell's law, expanding the cross products, and squaring both sides, we get

$$\frac{r_x^2 y_1^2}{r_x^2+(y_1-r_y)^2} = \left(\frac{n_2}{n_1}\right)^2 \frac{(r_x y_2 - r_y x_2)^2}{(x_2-r_x)^2+(y_2-r_y)^2} \tag{11.39}$$

Next, expand the squares, substitute $r_y^2 = r^2 - r_x^2$, and collect terms to get

$$\frac{c_1 r_x^2}{c_2+c_3 r_y} = \frac{c_4 r_x^2 + c_5 r_y r_x + c_6}{c_7 + c_8 r_x + c_9 r_y} \tag{11.40}$$

where $c_n = \left\{(n_1^2/n_2^2)y_1^2, r^2+y_1^2, -2y_1, y_2^2-x_2^2, -2x_2y_2, r^2x_2^2, x_2^2+y_2^2+r^2, -2x_2, -2y_2\right\}, n = 1...9$. To proceed, multiply through by the denominators, substitute r_y^2 again where it occurs, and collect r_y on one side of the equation, then

$$b_1 r_x^3 + b_2 r_x^2 + b_3 r_x + b_4 = (b_5 r_x^2 + b_6 r_x + b_7) r_y \tag{11.41}$$

where $b_n = \left\{c_3c_5 + c_1c_8, c_1c_7 - c_2c_4, -c_3c_5r^2, -c_2c_6, c_3c_4 - c_1c_9, c_2c_5, c_3c_6\right\}, n = 1...7$. Squaring both sides once more, substituting r_y^2 a third time, this can be written as a sixth degree polynomial in r_x as

$$\sum_{n=0}^{6} a_n r_x^n = 0 \tag{11.42}$$

$$a_6 = b_1^2 + b_5^2 \tag{11.43}$$

$$a_5 = 2b_1b_2 + 2b_5b_6 \tag{11.44}$$

$$a_4 = b_2^2 - b_5^2 r^2 + 2b_7b_5 + b_6^2 + 2b_1b_3 \tag{11.45}$$

$$a_3 = -2b_5b_6r^2 + 2b_1b_4 + 2b_2b_3 + 2b_6b_7 \tag{11.46}$$

$$a_2 = b_3^2 - b_6^2r^2 + b_7^2 - 2b_5b_7r^2 + 2b_2b_4 \tag{11.47}$$

$$a_1 = -2b_6b_7r^2 + 2b_3b_4 \tag{11.48}$$

$$a_0 = b_4^2 - b_7^2r^2 \tag{11.49}$$

Once the roots are computed, they are sifted to find the valid solution, because squaring twice created extraneous solutions. Roots with a non-zero imaginary part are rejected. Then r_y is computed and only solutions that satisfy Snell's law at the boundary are kept. In the event of a tie, the refraction point that gives the smallest electrical distance is kept, consistent with the principle of least action. Finally, we check that $\theta_1 < \pi/2$ to make sure the ray does not pass through the circle twice (this angle is computed with the dot product of $\mathbf{r}$ and $\mathbf{v}_1$ to safely handle $\pm\hat{x}$ solutions).

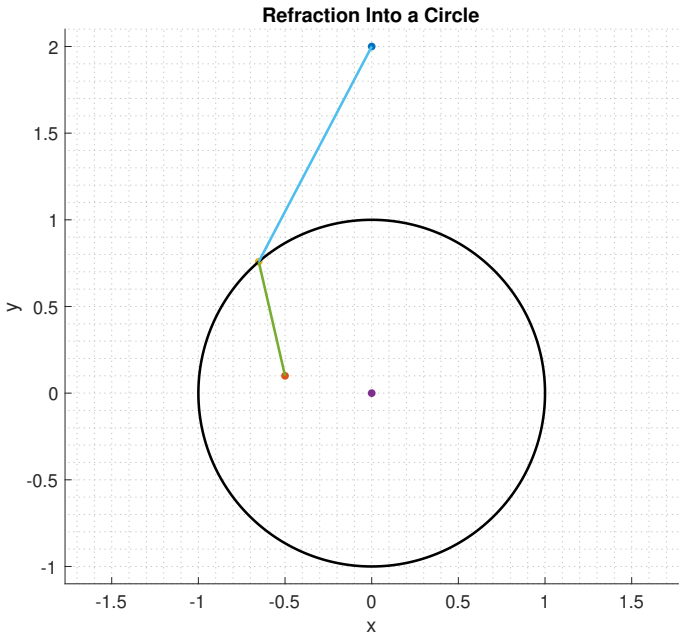


Figure 11.7: Refraction into a circle between two points. Exterior index of refraction $n_1 = 1$, interior index of refraction $n_2 = 2$, and radius $r = 1$.

The routine `refractionCircle` takes as input the coordinates of an arbitrary exterior point, $\mathbf{r}_1$, an arbitrary interior point, $\mathbf{r}_2$, index of refractions n_1, n_2 and radius r, and returns the coordinates of the refraction point on the circle, the incident and transmission angles θ_1 and θ_2 in radians, and the magnitudes of the vectors $\mathbf{v}_1$ and $\mathbf{v}_2$. It first rotates the points to align them with the y axis. It returns `nan` if no solution exists. It also assumes n_1 and n_2 are real. Because this relies on `roots` it is not vectorized.

```
function [rx ry theta1 theta2 thetal v1mag v2mag] = refractionCircle(x1,y1,x2,y2,n1,n2,r)
% Refraction through a circular interface
% assumes n1, n2 are real
%
% x1,y1     Coordinates of exterior point, r1, centered on circle
% x2,y2     Coordinates of interior point, r2, centered on circle
% n1        Index of refraction for exterior
% n2        Index of refraction for interior
% r         Radius circle
%
% rx,ry         Coordinates of refraction point
% theta1        Incidence angle
% theta2        Transmission angle
% thetal        Look angle measured from perpendicular
% v1mag, v2mag  Lengths of the vectors from each point to the refraction point

if sqrt(x1^2 + y1^2) < r
    error('exterior point is interior')
end
if sqrt(x2^2 + y2^2) > r
    error('interior point is exterior')
end
n1 = real(n1);
```

```
n2 = real(n2);
% rotate points to align r_1 with y-axis
theta_rot = atan2(y1,x1);
theta_rot = -(theta_rot - pi/2);
R = [cos(theta_rot) -sin(theta_rot);
       sin(theta_rot) cos(theta_rot)];
r1rot = R*[x1; y1];
x1 = r1rot(1);
y1 = r1rot(2);
r2rot = R*[x2; y2];
x2 = r2rot(1);
y2 = r2rot(2);

% compute c coefficients
c_1 = (n1^2/n2^2)*y1^2;
c_2 = r^2 + y1^2;
c_3 = -2*y1;
c_4 = y2^2 - x2^2;
c_5 = -2*x2*y2;
c_6 = r^2*x2^2;
c_7 = x2^2 + y2^2 + r^2;
c_8 = -2*x2;
c_9 = -2*y2;

% compute b coefficients
b_1 = c_3*c_5 + c_1*c_8;
b_2 = c_1*c_7 - c_2*c_4;
b_3 = -c_3*c_5*r^2;
b_4 = -c_2*c_6;
b_5 = c_3*c_4 - c_1*c_9;
b_6 = c_2*c_5;
b_7 = c_3*c_6;

% compute a coefficients
a_6 = b_1^2 + b_5^2;
a_5 = 2*b_1*b_2 + 2*b_5*b_6;
a_4 = b_2^2 - b_5^2*r^2 + 2*b_7*b_5 + b_6^2 + 2*b_1*b_3;
a_3 = -2*b_5*b_6*r^2 + 2*b_1*b_4 + 2*b_2*b_3 + 2*b_6*b_7;
a_2 = b_3^2 - b_6^2*r^2 + b_7^2 - 2*b_5*b_7*r^2 + 2*b_2*b_4;
a_1 = -2*b_6*b_7*r^2 + 2*b_3*b_4;
a_0 = b_4^2 - b_7^2*r^2;

% solve for the roots
coef = [a_6 a_5 a_4 a_3 a_2 a_1 a_0];
rt = roots(coef);

% reject roots with imaginary component
ind = (imag(rt)==0);
rx = rt(ind);

% compute ry
ry = sqrt(r^2 - rx.^2);

% compute vectors for remaining solutions
Nr = length(rx);
rr = [rx ry zeros(Nr,1)];
r1 = repmat([0 y1 0],Nr,1);
r2 = repmat([x2 y2 0],Nr,1);
v1 = r1 - rr;
v2 = r2 - rr;
rmag = sqrt(sum(rr.^2,2));
v1mag = sqrt(sum(v1.^2,2));
v2mag = sqrt(sum(v2.^2,2));
sint1 = cross(rr,v1,2)./v1mag./rmag;
sint2 = cross(-rr,v2,2)./v2mag./rmag;
sint1 = sint1(:,3);
sint2 = sint2(:,3);
```

```
% compute theta1 and theta2 via dot product
cost1 = dot(rr,v1,2)./v1mag./rmag;
theta1 = acos(cost1);
cost2 = dot(-rr,v2,2)./v2mag./rmag;
theta2 = acos(cost2);

% see which solutions satisfy Snell's law
test = n1*sint1 - n2*sint2;
ind = find(abs(test)<1e-11);

% ...of these, which has the smallest electrical length
eleclength = n1*v1mag(ind)  + n2*v2mag(ind);
[val ind2] = min(eleclength);
ind = ind(ind2);

% if a solution remains, check if theta1 is less than pi/2
if ~isempty(ind) && (theta1(ind) <= pi/2)
    rx = rx(ind);
    ry = ry(ind);
    theta1 = theta1(ind);
    theta2 = theta2(ind);
    thetal = atan2(rx,y1-ry);
    v1mag = v1mag(ind);
    v2mag = v2mag(ind);

    % rotate solution back
    rrot = R'*[rx; ry];
    rx = rrot(1);
    ry = rrot(2);
else
    rx = nan;
    ry = nan;
    theta1 = nan;
    theta2 = nan;
    thetal = nan;
    v1mag = nan;
    v2mag = nan;
end
```

11.5.2 Refraction Point - Circular Interface - Iterative

While the solution for refraction through a circular interface in the previous section is exact, the code could not be vectorized because it required the roots of a sixth degree polynomial. However, like the planar interface, a vectorized iterative solution can be derived. Its validity is restricted to mild refraction and small angles but it is lightning fast when it works. The primary application for this is subsurface SAR focusing in, for instance, radar sounding through ice at planetary bodies, where the body curvature needs to be included in the refraction for a large number of subsurface focal points.

Using the geometry in Figure 11.6, define:

$$h = y_1 - r \tag{11.50}$$

$$d = r - y_2 \tag{11.51}$$

Using the fact that

$$\sin\theta_1 = \frac{r+h}{r}\sin\theta_l \tag{11.52}$$

we can build an iteration as

$$
\begin{aligned}
\text{Initialize:} \quad \theta_l &\leftarrow \arctan\left(\frac{x_2}{h+d}\right) \\
\text{Iterate:} \quad \theta_1 &\leftarrow \arcsin\left(\frac{r+h}{r}\sin\theta_l\right) \\
\theta_2 &\leftarrow \arcsin\left(\frac{n_2}{n_1}\sin\theta_1\right) && (11.53) \\
\theta_p &\leftarrow \theta_2 - (\theta_1 - \theta_l) && (11.54) \\
(r_x, r_y) &\leftarrow \text{ComputeIntersection}(\theta_p) && (11.55) \\
\theta_l &\leftarrow \arctan\left(\frac{r_x}{h+r-r_y}\right)
\end{aligned}
$$

where θ_p is the angle from the perpendicular at the refraction point to the interior point, and ComputeIntersection(θ_p) computes the point at which a line from the interior point with slope corresponding to θ_p intersects the circle.

This iteration starts with the straight ray approximation for the look angle θ_l from which it computes the incident angle on the circle, θ_1. With this, the transmission angle, θ_2, is computed via Snell's law. Next, θ_p is the portion of θ_2 relative to the perpendicular. Using this, we project a line from the interior point to intersect the circle. The slope of the line corresponds to the angle θ_p. With the new point on the circle, the look angle is computed again and the process repeats. This is again a Newton iteration of the transcendental solution for one of the variables. As before, the solution is not the most robust and should be used when the refraction is mild.

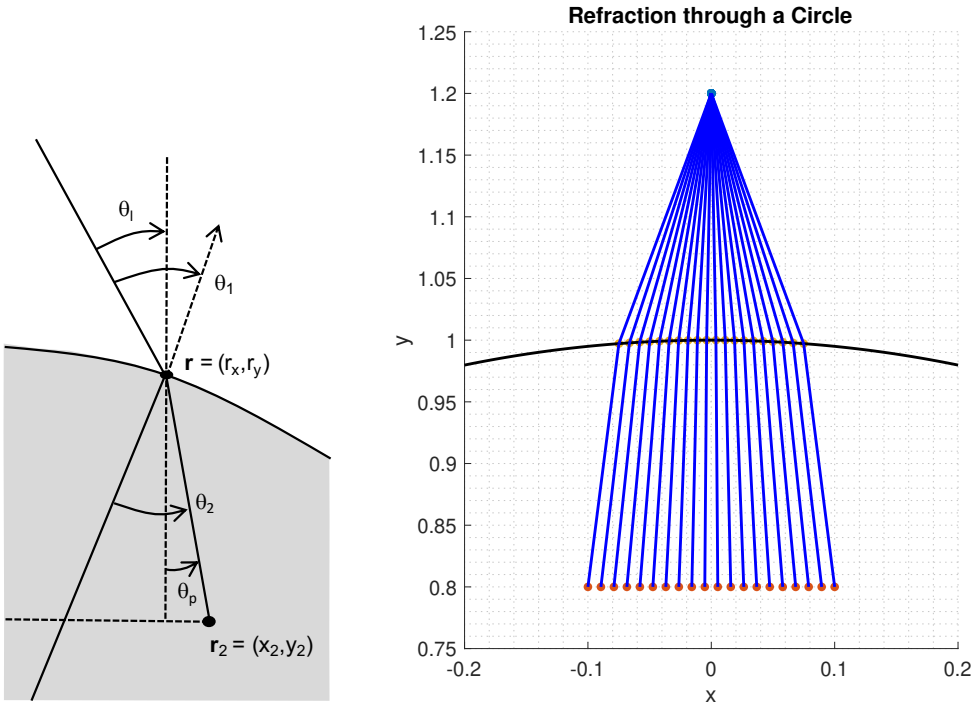


Figure 11.8: Left: Angles at the refraction point for iterative solution. Right:Refraction into a circle between multiple points using the vectorized iterative solution. Exterior index of refraction $n_1 = 1$, interior index of refraction $n_2 = 2$, and radius $r = 1$.

Intersection of Circle and Line Given a line $y = mx + b$ and a circle $(x-p)^2 + (y-q)^2 = r^2$, the point of intersection (X, Y) is found by substituting the first equation into the second and solving the resulting quadratic for x:

$$\begin{aligned} A &= m^2 + 1 \\ B &= 2(mb - mq - p) \\ C &= q^2 - r^2 + p^2 - 2bq + b^2 \\ X &= \frac{-B \pm \sqrt{B^2 - 4AC}}{2A} \\ Y &= mX + b \end{aligned}$$

ComputeIntersection(θ_p) Let the center of the circle be the origin $(p, q) = (0, 0)$. The line with slope corresponding to angle θ_p passing though the interior point at (x_2, y_2) is given by

$$y = -\cot\theta_p(x - x_2) + y_2 \tag{11.56}$$

The parameters used in the equations above for the intersection of the line and circle are therefore

$$\begin{aligned} m &= -\cot\theta_p \\ b &= x_2\cot\theta_p + y_2 \\ A &= m^2 + 1 \\ B &= 2mb \\ C &= -r^2 + b^2 \end{aligned}$$

For this problem, the choice of quadratic root is determined by the sign of x_2. Furthermore, if the y coordinate of the refraction point is negative at any point in the iteration, then the other root is chosen. Points that are collinear with the origin and the source, where there is no refraction, need to be handled separately. Finally, in this geometry, the look angle is given by

$$\tan\theta_l = \frac{r_x}{h + r - r_y} \tag{11.57}$$

The routine `refractionCircleIterative` is a vectorized implementation of the routine above. It takes an input pairs of $\mathbf{r}_1$, $\mathbf{r}_2$, the index of refraction n_1, n_2, radius r, and number of iterations, `nit`, and returns the refraction points, $\mathbf{r}$, incident and transmission angles θ_1, θ_2, θ_l, and the lengths of the vectors $\mathbf{v}_1$ and $\mathbf{v}_2$. The point arrays can be any size. This assumes n_1 and n_2 are real. This should only be used in its present form for large radii, and depths that are less than or equal to the heights. Set `nit` to be 10 or larger, it defaults to 40. The algorithm rotates points to align the exterior points to the y axis, to be consistent with the solution as written, then rotates them back.

```
function [rx ry theta1 theta2 thetal v1mag v2mag] = refractionCircleIterative(x1,y1,x2,y2,n1,n2,r,nit)
% Refraction through a circular interface, iterative solution
% assumes n1, n2 are real
%
% x1,y1     Coordinates of exterior point, r1, centered on circle
% x2,y2     Coordinates of interior point, r2, centered on circle
% n1        Index of refraction for exterior
% n2        Index of refraction for interior
% r         Radius circle
% nit       Number of iterations
%
% rx,ry         Coordinates of refraction point
% theta1        Incidence angle
% theta2        Transmission angle
% thetal        Look angle measured from perpendicular
% v1mag, v2mag  Lengths of the vectors from each point to the refraction point

sz =size(x1);
n1 = real(n1);
```

```
n2 = real(n2);

% rotate points to align r_1 with y-axis
x1 = x1(:);
y1 = y1(:);
N = length(x1);
for n=1:N,
    theta_rot = atan2(y1(n),x1(n));
    theta_rot = -(theta_rot - pi/2);
    R = [cos(theta_rot) -sin(theta_rot);
       sin(theta_rot) cos(theta_rot)];
    r1rot = R*[x1(n)'; y1(n)'];
    x1(n) = r1rot(1);
    y1(n) = r1rot(2);
    r2rot = R*[x2(n); y2(n)];
    x2(n) = r2rot(1);
    y2(n) = r2rot(2);
end

% aux parameters
h = y1 - r;
d = r - y2;

% default number o iterations
if nargin == 7
    nit = 40;
end

% initialized iterations
thetal = atan2(x2,h + d);

% find points along radial line
ind = find(thetal==0);

for n=1:nit
    % compute incident, transmission and point angles
    sin_theta1 = asin(((r + h)./r).*sin(thetal));
    theta1 = asin(sin_theta1);
    theta2 = asin(n1./n2.*sin_theta1);
    theta_p = theta2 - (theta1 - thetal);

    % solve for intersection point
    m = -cot(theta_p);
    b = x2.*cot(theta_p) + y2;
    A = m.^2 + 1;
    B = 2*m.*b;
    C = -r.^2 + b.^2;
    rx = (-B-sign(x2).*sqrt(B.^2 - 4*A.*C))./(2*A);
    ry = (m.*rx + b);

    % if refraction point is on wrong side of sphere, use other solution
    ind2 = (ry<y2);
    rx(ind2) = (-B(ind2) + sign(x2(ind2)).*sqrt(B(ind2).^2 - 4*A(ind2).*C(ind2)))./(2*A(ind2));
    ry(ind2) = (m(ind2).*rx(ind2) + b(ind2));

    % update look angle
    thetal = atan2(rx,h + r - ry);
end

% take care of radial points
rx(ind) = 0;
ry(ind) = r;
thetal(ind) = 0;
theta1(ind) = 0;
theta2(ind) = 0;
```

```
% compute vectors for remaining solutions
v1 = [(x1-rx), (y1-ry)];
v2 = [(x2-rx), (y2-ry)];
v1mag = sqrt(sum(v1.^2,2));
v2mag = sqrt(sum(v2.^2,2));

% rotate solution back
for n=1:N,
    theta_rot = atan2(y1(n),x1(n));
    theta_rot = -(theta_rot - pi/2);
    R = [cos(theta_rot) -sin(theta_rot);
       sin(theta_rot) cos(theta_rot)];
    rrrot = R'*[rx(n); ry(n)];
    rx(n) = rrrot(1);
    ry(n) = rrrot(2);
end

rx=reshape(rx,sz);
ry=reshape(ry,sz);
theta1=reshape(theta1,sz);
theta2=reshape(theta2,sz);
thetal=reshape(thetal,sz);
v1mag=reshape(v1mag,sz);
v2mag=reshape(v2mag,sz);
```

11.6 Refraction Point - Sphere

We can adapt the solution for the refraction point over a circle to that for an arbitrary interior and exterior points over a sphere. Let $\mathbf{r}_1$ be the exterior point, $\mathbf{r}_2$ be the interior point, and $\mathbf{r}$ be the point of refraction on the sphere. The sphere has radius r centered at the origin. Define a new coordinate system $[\hat{u}, \hat{v}, \hat{w}]$ in the $\mathbf{r}_1$-$\mathbf{r}_2$ plane such that

$$\hat{v} = \hat{r}_1 \tag{11.58}$$

$$\hat{w} = \frac{\mathbf{r}_2 \times \hat{v}}{|\mathbf{r}_2 \times \hat{v}|} \tag{11.59}$$

$$\hat{u} = \hat{v} \times \hat{w} \tag{11.60}$$

The exterior and interior points are represented in this frame as

$$\mathbf{r}_1' = |\mathbf{r}_1|\hat{v} \tag{11.61}$$

$$\mathbf{r}_2' = (\mathbf{r}_2 \cdot \hat{u})\,\hat{u} + (\mathbf{r}_2 \cdot \hat{v})\,\hat{v} \tag{11.62}$$

The coordinates of $\mathbf{r}_1'$ and $\mathbf{r}_2'$ can now be used as inputs to either of the two circular refraction routines from Section 11.5.1 or 11.5.2. The magnitude of $\mathbf{r}_1'$ in the $\hat{v}$ direction corresponds to the y-coordinate of the exterior point in both routines, while $(\mathbf{r}_2 \cdot \hat{u})$ and $(\mathbf{r}_2 \cdot \hat{v})$ correspond to the x and y coordinates, respectively, of the interior point in both routines. Once solved, the x and y coordinates of the refraction point on the 2D circle, r_x and r_y, are mapped back to the original 3D frame as

$$\mathbf{r} = r_x\hat{u} + r_y\hat{v} \tag{11.63}$$

The routine `refractionSphere` computes the refraction through a sphere that is centered at the origin. It takes as input the Cartesian coordinates of pairs of exterior and interior points of any array size, the index of refraction for the exterior and interior, n_1, n_2, respectively, and the radius of the sphere r. The outputs are the coordinates of the refraction point on the sphere, incident angle, transmission angle, and look angle which measured from the radial line of the exterior point, as well as the two path lengths inside and outside the sphere. The routine transforms the input coordinates using the mapping above in order to use the circle refraction algorithms, `refractionCircle` or `refractionCircleIterative`. The routine

defaults to the direct circle refraction solution which loops over each pair of points. The direction solution can also be chosen using the string switch `'dir'`. Use string switch `'it'` to select the vectorized iterative circle refraction solution which can also take an optional input argument for the number of iterations.

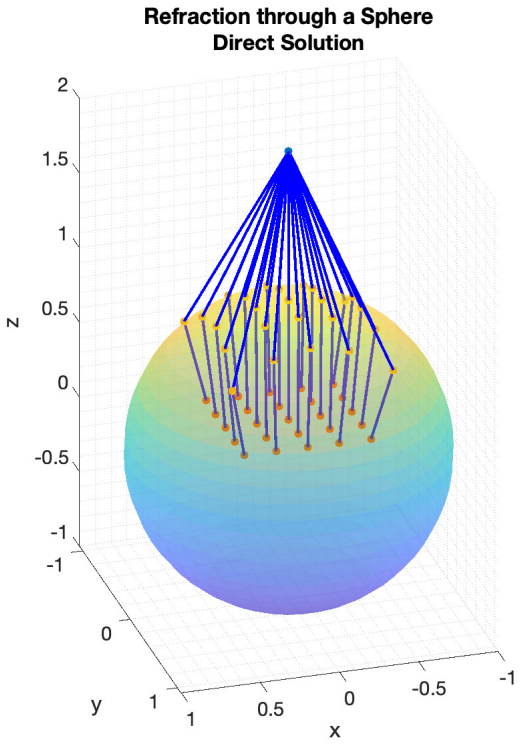


Figure 11.9: Refraction into a sphere between multiple points. Exterior index of refraction $n_1 = 1$, interior index of refraction $n_2 = 2$, and sphere radius $r = 1$.

```
function [rx ry rz theta1 theta2 thetal v1mag v2mag] = refractionSphere(x1,y1,z1,x2,y2,z2,n1,n2,r,type,nit)
% Refraction through a spherical interface, assumes n1, n2 are real.
%
% x1,y1,z1      Coordinates of exterior point, r1, centered on circle
% x2,y2,z2      Coordinates of interior point, r2, centered on circle
% n1            Index of refraction for exterior
% n2            Index of refraction for interior
% r             Radius circle
% type          [optional] Defaults to direct solution.
%               Use 'dir' for direct solution, 'it' for iterative.
% nit           [optional] Number of iterations when using 'it' option
%
% rx,ry,rz      Coordinates of the refraction point
% theta1        Incidence angle
% theta2        Transmission angle
% thetal        Look angle
% v1mag, v2mag  Lengths of the vectors from each point to the refraction point
%
% Dependencies: refractionCircle, refractionCircleIterative

sz = size(x1);
n1 = real(n1);
n2 = real(n2);

% columnize the inputs
r1 = [x1(:) y1(:) z1(:)];
r2 = [x2(:) y2(:) z2(:)];
```

```
N = length(x1(:));

% project the points in the r1-r2 plane
% v_hat
r1mag = sqrt(sum(r1.^2,2));
v_hat = r1./repmat(r1mag,1,3);
% w_hat
r2xvhat = cross(r2,v_hat,2);
% check for zero values (interoir and exterior point are radially aligned)
ind = find(sum(abs(r2xvhat),2)==0);
% in these cases, create an arbitrary cross product by permuting r1
tmp = [r1(ind,2) r1(ind,3) r1(ind,1)];
r2xvhat(ind,:) = tmp;
w_hat = r2xvhat./repmat(sqrt(sum(r2xvhat.^2,2)),1,3);
% u_hat
u_hat = cross(v_hat,w_hat,2);

% in the circle frame:
% y coordinte of exterior point
c1x = zeros(N,1);
c1y = r1mag;
% x coordinate of interior point
c2x = dot(r2,u_hat,2);
c2y = dot(r2,v_hat,2);

% initialize outputs
rx = zeros(N,1);
ry = zeros(N,1);
theta1 = zeros(N,1);
theta2 = zeros(N,1);
thetal = zeros(N,1);
v1mag = zeros(N,1);
v2mag = zeros(N,1);

if nargin >= 10 && strcmp(type,'dir') || nargin >= 10 && isempty(type) % loop over direct solution
    for n=1:N,
        [rxt ryt t1 t2 tl v1m v2m] = refractionCircle(c1x(n),c1y(n),c2x(n),c2y(n),n1,n2,r);
        rx(n) = rxt;
        ry(n) = ryt;
        theta1(n) = t1;
        theta2(n) = t2;
        thetal(n) = tl;
        v1mag(n) = v1m;
        v2mag(n) = v2m;
    end
elseif nargin >= 10 && strcmp(type,'it') % vectorized iterative solution
    if ~isempty(nit)
        [rx ry theta1 theta2 thetal v1mag v2mag] = refractionCircleIterative(c1x,c1y,c2x,c2y,n1,n2,r,nit);
    else
        [rx ry theta1 theta2 thetal v1mag v2mag] = refractionCircleIterative(c1x,c1y,c2x,c2y,n1,n2,r);
    end
else
    error('bad inputs')
end

% project refraction point back to 3D
rr = repmat(rx,1,3).*u_hat + repmat(ry,1,3).*v_hat;

% reshape to original size of inputs
rx=reshape(rr(:,1),sz);
ry=reshape(rr(:,2),sz);
rz=reshape(rr(:,3),sz);
theta1=reshape(theta1,sz);
theta2=reshape(theta2,sz);
thetal=reshape(thetal,sz);
v1mag=reshape(v1mag,sz);
v2mag=reshape(v2mag,sz);
```

Chapter 12

Utilities

This chapter contains routines for coordinate transformations, special functions and some miscellaneous but useful computations. Routines for special functions can be found elsewhere, however we give some spherical Bessel function routines, because Matlab does not have them, and a collection of routines for different variations of Legendre polynomials and derivatives that are needed for computing spherical wave functions. Finally, miscellaneous routines include things like FFT frequency generation, vectorized solutions for cubic and quartic roots, and simple ways for generating points a sphere.

12.1 Coordinate Transformations

Table 12.1 contains coordinates transformations between Cartesian, cylindrical, and spherical coordinates and vector fields, [48]. The routines in Table 12.1 overload the built-in Matlab functions of the same names in order to use θ, ϕ orderings and definitions consistent with most scattering textbooks (physics convention, rather than mathematics convention which is what Matlab uses). The same routines transform between either coordinate points or vector fields. For example, transforming Cartesians points (x, y, z) to spherical coordinates (r, θ, ϕ) is done as

```
[r, th, phi] = cart2sph(x,y,z)
```

The vector transforms always needs the coordinates of the input vector field. For example, transforming a Cartesian vector field (A_x, A_y, A_z) located at points (x, y, z) to spherical vector components (A_r, A_θ, A_ϕ) is done as

```
[Ar, Ath, Aphi] = cart2sph(x,y,z,Ax,Ay,Az)
```

Input arrays can be any equal size and $\tan^{-1}$ is always computed with `atan2`. Unit vectors can be created by setting one of the input vector components equal to 1 and the others to 0. For example, the Cartesian unit vectors at a point (x, y, z) expressed in spherical coordinates are computed as

```
[x_r, x_th, x_phi] = cart2sph(x,y,z,1,0,0)
[y_r, y_th, y_phi] = cart2sph(x,y,z,0,1,0)
[z_r, z_th, z_phi] = cart2sph(x,y,z,0,0,1)
```

Table 12.1: Coordinate Transforms

Routine	Point Transforms	Vector Field Transforms
`cart2cyl`	$\rho = \sqrt{x^2 + y^2}$ $\phi = \tan^{-1}(y/x)$ $z = z$	$A_\rho = A_x \cos\phi + A_y \sin\phi$ $A_\phi = -A_x \sin\phi + A_y \cos\phi$ $A_z = A_z$
`cyl2cart`	$x = \rho \cos\phi$ $y = \rho \sin\phi$ $z = z$	$A_x = A_\rho \cos\phi - A_\phi \sin\phi$ $A_y = A_\rho \sin\phi + A_\phi \cos\phi$ $A_z = A_z$
`cart2sph`	$r = \sqrt{x^2 + y^2 + z^2}$ $\theta = \tan^{-1}(\sqrt{x^2 + y^2}/z)$ $\phi = \tan^{-1}(y/x)$	$A_r = A_x \sin\theta\cos\phi + A_y \sin\theta \sin\phi + A_z \cos\theta$ $A_\theta = A_x \cos\theta\cos\phi + A_y \cos\theta \sin\phi - A_z \sin\theta$ $A_\phi = -A_x \sin\phi + A_y \cos\phi$
`sph2cart`	$x = r \sin\theta\cos\phi$ $y = r \sin\theta \sin\phi$ $z = r \cos\theta$	$A_x = A_r \sin\theta\cos\phi + A_\theta \cos\theta\cos\phi - A_\phi \sin\phi$ $A_y = A_r \sin\theta\sin\phi + A_\theta \cos\theta\sin\phi + A_\phi \cos\phi$ $A_z = A_r \cos\theta - A_\theta \sin\theta$
`cyl2sph`	$r = \sqrt{\rho^2 + z^2}$ $\theta = \tan^{-1}(\rho/z)$ $\phi = \phi$	$A_r = A_\rho \sin\theta + A_z \cos\theta$ $A_\theta = A_\rho \cos\theta - A_z \sin\theta$ $A_\phi = A_\phi$
`sph2cyl`	$\rho = r \sin\theta$ $\phi = \phi$ $z = r \cos\theta$	$A_\rho = A_r \sin\theta + A_\theta \cos\theta$ $A_\phi = A_\phi$ $A_z = A_r \cos\theta - A_\theta \sin\theta$

Routine `cart2cyl`

```
function [out1 out2 out3] = cart2cyl(x,y,z,Ax,Ay,Az)
% Cartesian to cylindrical coordinate transformation.  Angles
% in radians.
%
% x,y,z:        Cartesian points
% Ax,Ay,Az:     Cartesian vector components
%
% out1,out2,out3
%       rho,phi,z:    Cylindrical points
%    or
%       Arho,Aphi,Az: Cylindrical vector components

if checkargs(nargout,nargin)
    return
end
if nargin == 3
    out1 = sqrt(x.^2 + y.^2);
    out2 = atan2(y,x);
    out3 = z;
end
if nargin == 6
    phi = atan2(y,x);
    out1 = Ax.*cos(phi) + Ay.*sin(phi);
    out2 = -Ax.*sin(phi) + Ay.*cos(phi);
    out3 = Az;
end
```

Routine cyl2cart

```
function [out1 out2 out3] = cyl2cart(rho,phi,z,Arho,Aphi,Az)
% Cylindrical to Cartesian coordinate transformation.  Angles
% in radians.
%
% rho,phi,z:    Cylindrical points
% Arho,Aphi,Az: Cylindrical vector components
%
% out1,out2,out3
%       x,y,z:        Cartesian points
%   or
%       Ax,Ay,Az:     Cartesian vector components

if checkargs(nargout,nargin)
    return
end
if nargin == 3
    out1 = rho.*cos(phi);
    out2 = rho.*sin(phi);
    out3 = z;
end
if nargin == 6
    out1 = Arho.*cos(phi) - Aphi.*sin(phi);
    out2 = Arho.*sin(phi) + Aphi.*cos(phi);
    out3 = Az;
end
```

Routine cart2sph

```
function [out1 out2 out3] = cart2sph(x,y,z,Ax,Ay,Az)
% Cartesian to spherical coordinate transformation.  Angles
% in radians.
%
% x,y,z:        Cartesian points
% Ax,Ay,Az:     Cartesian vector components
%
% out1,out2,out3
%       r,theta,phi:   Spherical points
%    or
%       Ar,Ath,Aphi: Spherical vector components

if checkargs(nargout,nargin)
    return
end
if nargin == 3
    out1 = sqrt(x.^2 + y.^2 + z.^2);
    out2 = atan2(sqrt(x.^2 + y.^2),z);
    out3 = atan2(y,x);
end
if nargin == 6
    th = atan2(sqrt(x.^2 + y.^2),z);
    phi = atan2(y,x);
    out1 = Ax.*sin(th).*cos(phi) + Ay.*sin(th).*sin(phi) + Az.*cos(th);
    out2 = Ax.*cos(th).*cos(phi) + Ay.*cos(th).*sin(phi) - Az.*sin(th);
    out3 = -Ax.*sin(phi) + Ay.*cos(phi);
end
```

Routine sph2cart

```
function [out1 out2 out3] = sph2cart(r,th,phi,Ar,Ath,Aphi)
% Cartesian to spherical coordinate transformation.  Angles
% in radians.
%
% rho,phi,z:        Spherical points
% Ar,Ath,Aphi:      Spherical vector components
%
% out1,out2,out3
```

```
%       x,y,z:        Cartesian points
%    or
%       Ax,Ay,Az:     Cartesian vector components

if checkargs(nargout,nargin)
    return
end
if nargin == 3
    out1 = r.*sin(th).*cos(phi);
    out2 = r.*sin(th).*sin(phi);
    out3 = r.*cos(th);
end
if nargin == 6
    out1 = Ar.*sin(th).*cos(phi) + Ath.*cos(th).*cos(phi) - Aphi.*sin(phi);
    out2 = Ar.*sin(th).*sin(phi) + Ath.*cos(th).*sin(phi) + Aphi.*cos(phi);
    out3 = Ar.*cos(th) - Ath.*sin(th);
end
```

Routine cyl2sph

```
function [out1 out2 out3] = cyl2sph(rho,phi,z,Arho,Aphi,Az)
% Cyindrical to spherical coordinate transformation.  Angles
% in radians.
%
% rho,phi,z:        Cyindrical points
% Arho,Aphi,Az:     Cyindrical vector components
%
% out1,out2,out3
%       r,th,phi:       Spherical points
%    or
%       Ar,Ath,Aphi:        Spherical vector components

if checkargs(nargout,nargin)
    return
end
if nargin == 3
    out1 = sqrt(rho.^2 + z.^2);
    out2 = atan2(rho,z);
    out3 = phi;
end
if nargin == 6
    th = atan2(rho,z);
    out1 = Arho.*sin(th) + Az.*cos(th);
    out2 = Arho.*cos(th) - Az.*sin(th);
    out3 = Aphi;
end
```

Routine sph2cyl

```
function [out1 out2 out3] = sph2cyl(r,th,phi,Ar,Ath,Aphi)
% Spherical to cylinder coordinate transformation.  Angles
% in radians.
%
% r,th,phi:        Spherical points
% Ar,Ath,Aphi:      Spherical vector components
%
% out1,out2,out3
%       rho,phi,z:            Cylinder points
%    or
%       Arho,Aphi,Az:         Cylinder vector components

if checkargs(nargout,nargin)
    return
end
if nargin == 3
    out1 = r.*sin(th);
    out2 = phi;
    out3 = r.*cos(th);
```

```
end
if nargin == 6
    out1 = Ar.*sin(th) + Ath.*cos(th);
    out2 = Aphi;
    out3 = Ar.*cos(th) - Ath.*sin(th);
end
```

Helper Routine checkargs

```
function [yn] = checkargs(b,a)
% Coordinate transforms helper function

yn = ~((a == 3 && b == 3) || (a == 6 && b == 3));
if yn
    disp('not enough input/ouput arguments')
end
```

12.2 Spherical Bessel Functions

Matlab does not have built-in routines for spherical Bessel functions. Below are several variants of spherical Bessel functions and derivatives encountered in scattering problems.

12.2.1 Sombrero function: jinc(x)

The jinc(x) function, or Sombrero function, is defined:

$$\text{jinc}(x) = \frac{J_1(x)}{x} \tag{12.1}$$

where small arguments are computed with the Taylor series

$$\text{jinc}(x) \approx \frac{1}{2} - \frac{x^2}{16} + O(x^4) \tag{12.2}$$

```
function jc = jinc(x)
% jinc (Sombrero) function
%
% jc:   jinc function: jinc(x) = J_1(x)/x

jc = besselj(1,x)./x;
thresh = 1e-5;
jc((x<thresh)) = 1/2-x((x<thresh)).^2/16;
```

12.2.2 Spherical Bessel function: $j_n(x)$

The spherical Bessel function is

$$j_n(x) = \sqrt{\frac{\pi}{2x}} J_{n+1/2}(x) \tag{12.3}$$

where

$$\lim_{x\to 0} j_n(x) = \begin{cases} 1 - \dfrac{x^2}{6} + O(x^4) & n = 0 \\ 0, & n \geq 1 \end{cases} \tag{12.4}$$

The routine `sbesselj` returns the spherical Bessel function for matching arrays of n and x in the format of `besselj`. It uses the Taylor series near $x = 0$ for $n = 0$, and substitutes 0 when $x = 0$ for $n > 1$.

```
function j = sbesselj(n,x)
% Spherical Bessel function
%
%   j_n(x) = sqrt(pi/(2x)) J_(n+1/2)(x)
%
% n:    order n
% x:    independent variable
%
% j:    spherical bessel function order n evaluated at x

j = sqrt(pi./(2*x)).*besselj(n+0.5,x);
thresh = 1e-5;
j(and((x<thresh),(n==0))) = 1 - x(and((x<thresh),(n==0))).^2/6;
j(and((x==0),(n~=0))) = 0;
```

12.2.3 Spherical Bessel function: $j_n(x)/x$

This variant of the spherical Bessel function has $j_n(x)$ divided by x:

$$\frac{j_n(x)}{x} = \frac{1}{x^{3/2}}\sqrt{\frac{\pi}{2}}J_{n+1/2}(x) \tag{12.5}$$

where

$$\lim_{x\to 0} j_n(x) = \begin{cases} \infty, & n = 0 \\ \frac{1}{3} - \frac{x^2}{30} + O(x^4), & n = 1 \\ 0, & n \geq 2 \end{cases} \tag{12.6}$$

This is computed in the routine `sbesselj2`, which takes the same inputs as `sbesselj`. We let the routine return `Inf` for $x = 0$ when $n = 0$. This variant is found in electromagnetic scattering when usually the monopole term ($n = 0$) is not needed.

```
function j = sbesselj2(n,x)
% Spherical Bessel function variant
%
%   j_n(x)/x
%
% n:    order n
% x:    independent variable
%
% j:    spherical bessel function order n evaluated at x, divided by x

j = sbesselj(n,x)./x;
thresh = 1e-5;
j(and((x<thresh),(n==1))) = 1/3 - x(and((x<thresh),(n==1))).^2/30;
j(and((x==0),(n>1))) = 0;
```

12.2.4 Spherical Bessel function derivative: $j_n'(x)$

The derivative of the spherical Bessel function with respect to x is given by the recurrence relation

$$j_n'(x) = -j_{n+1}(x) + \frac{n}{x}j_n(x) \tag{12.7}$$

This is computed in `sbesseljp` with `sbesselj` and `sbesselj2`, which takes care of $x = 0$.

```
function jp = sbesseljp(n,x)
% Spherical Bessel function derivative
%
%        j_n'(x) = -j_(n+1)(x) + n/x * j_n(x)
%
% n:    order n
% x:    independent variable
%
% jp:   spherical bessel function derivative
%
% Dependencies: sbesselj, sbesselj2

jp = -sbesselj(n+1,x) + n.*sbesselj2(n,x);
```

12.2.5 Spherical Bessel function derivative: $[xj_n(x)]'$

This variant occurs often enough in scattering to have its own function, `sbesseljp2`,

$$[xj_n(x)]' = j_n(x) + xj_n'(x) \tag{12.8}$$
$$= (1+n)j_n(x) - xj_{n+1}(x) \tag{12.9}$$

This variant is found in the Mie scattering solution for spheres.

```
function jp2 = sbesseljp2(n,x)
% Spherical Bessel function derivative, second type
%
%   (x j_n(x))' = j_n(x) + x j_n'(x)
%
% n:    order of bessel function
% x:    independent variable
%
% jp2:  spherical bessel function derivative variant
%
% Dependencies: sbesselj

jp2 = (1+n).*sbesselj(n,x) - x.*sbesselj(n+1,x);
```

12.2.6 Spherical Hankel function: $h_n^{(1)}(x)$

The spherical Hankel function is given by

$$h_n^{(1)}(x) = \sqrt{\frac{\pi}{2x}} H_{n+1/2}(x) \tag{12.10}$$

This is always irregular at the origin, and computed in the routine `sbesselh`

```
function h = sbesselh(n,x)
% Spherical Hankel function
%
%   h_n(x) = sqrt(pi/(2x)) H_(n+1/2)(x)
%
% n:    order n
% x:    independent variable
%
% h:    spherical Hankel function order n evaluated at x

h = sqrt(pi./(2*x)).*besselh(n+0.5,x);
```

12.2.7 Spherical Hankel function derivative: $h_n'^{(1)}(x)$

The spherical Hankel derivative is computed in the routine `sbesselhp`

$$h_n'^{(1)}(x) = -h_{n+1}^{(1)}(x) + \frac{n}{x} h_n^{(1)}(x) \tag{12.11}$$

```
function hp = sbesselhp(n,x)
% Spherical Hankel function derivative
%
%        h_n'(x) = -h_(n+1)(x) + n/x * h_n(x)
%
% n:    order n
% x:    independent variable
%
% hp:   spherical Hankel function derivative
%
% Dependencies: sbesselh

hp = -sbesselh(n+1,x) + (n./x).*sbesselh(n,x);
```

12.2.8 Spherical Hankel function derivative: $[xh_n^{(1)}(x)]'$

Similar to before, this variant is computed in the routine `sbesselhp2`

$$[xh_n^{(1)}(x)]' = h_n^{(1)}(x) + xh'^{(1)}_n(x) \tag{12.12}$$
$$= (1+n)h_n^{(1)}(x) - xh_{n+1}^{(1)}(x) \tag{12.13}$$

This variant is found in the Mie scattering solution for spheres.

```
function hp2 = sbesselhp2(n,x)
% Spherical Bessel function derivative, second type
%
%   (x h_n(x))' = h_n(x) + x h_n'(x)
%
% n:    order of bessel function
% x:    independent variable
%
% hp2:  spherical Hankel function derivative variant
%
% Dependencies: sbesselh

hp2 = (1+n).*sbesselh(n,x) - x.*sbesselh(n+1,x);
```

12.3 Legendre Polynomials

The routines in this section compute Legendre polynomials, normalized associated Legendre polynomials and their derivatives in formats useful to scattering. Associated Legendre polynomials are computed at both positive and negative m and automatically include the Condon-Shortley phase.

12.3.1 Legendre Polynomials, $P_l(x)$

The Legendre polynomials satisfy the equation

$$\frac{d}{dx}\left[(1-x^2)\frac{d}{dx}P_l(x)\right] + l(l+1)P_l(x) = 0 \tag{12.14}$$

for $-1 \leq x \leq 1$, with the following recurrence relation

$$(l+1)P_{l+1}(x) = (2l+1)xP_l(x) - lP_{l-1}(x) \tag{12.15}$$

The routine `legendrePl` returns $P_l(x)$ on a 2D array with degrees 0 to L along rows and evaluation points along columns.

```
function [Pl] = legendrePl(L,x)
% Legendre polynomial computed with:
%
% (l+1)P_{l+1}(x) = (2l+1)xP_l(x) - lP_{l-1}(x)
%
% L:    maximum degree L
% x:    evaluateion points -1 < x < 1
%
% Pl:   legendre polynomials evaluated for 0:L all x
%       Size (L+1) X length(x)

if L < 0
    error('bad degree')
end
x = x(:)';
ind = or((x < -1),(x > 1));
if ~isempty(ind)
    x(ind) = NaN;
end
Pl = zeros(L+1,length(x));
Pl(1,:) = 1;
if L == 0
    return
end
Pl(2,:) = x;
if L == 1
    return
end
for n=1:(L-1),
   c1 = (2*n+1)/(n+1);
   c2 = -n/(n+1);
   Pl(n+2,:) = c1*(x.*Pl(n+1,:)) + c2*Pl(n,:);
end
```

12.3.2 Legendre Polynomial Derivative, $d/dxP_l(x)$

The derivative of the Legendre polynomials satisfies either of the following recurrence relations

$$\frac{x^2-1}{l}\frac{d}{dx}P_l(x) = xP_l(x) - P_{l-1}(x) \tag{12.16}$$

$$(2l+1)P_l(x) = \frac{d}{dx}(P_{l+1}(x) - P_{l-1}(x)) \tag{12.17}$$

The first has a divide by zero when x is 1 or -1. Rearranging the second equation we get following recurrence

$$\frac{d}{dx}P_{l+1}(x) = (2l+1)P_l(x) + \frac{d}{dx}P_{l-1}(x) \tag{12.18}$$

which is safe on $x = [-1, 1]$. Initial conditions are the special cases $P_0'(x) = 0$ and $P_1'(x) = 1$, and the derivative has the feature that $P_l'(1) = l(l+1)/2$. The routine `legendrePlp` is like `legendrePl`: it returns $d/dxP_l(x)$ on a 2D array with degrees 0 to L along rows and evaluation points along columns.

```
function [Plp] = legendrePlp(L,x)
% Legendre polynomial derivative computed with:
%
% P'_{l+1}(x) = (2l+1)P_l(x) + P'_{l-1}(x)
%
% L:    maximum degree L
% x:    evaluateion points -1 < x < 1
%
% Plp:  legendre polynomials evaluated for 0:L all x
%       Size (L+1) X length(x)
%
% Dependencies: legendrePl

if L < 0
    error('bad degree')
end
x = x(:)';
ind = or((x < -1),(x > 1));
if ~isempty(ind)
    x(ind) = NaN;
end
Plp = zeros(L+1,length(x));
if L == 0
    return
end
Plp(2,:) = 1;
if L == 1
    return
end
Pl = legendrePl(L,x);
for n=2:L,
   Plp(n+1,:) = (2*(n-1)+1)*Pl(n,:) + Plp(n-1,:);
end
```

12.3.3 Normalized Associated Legendre Polynomials, $\widetilde{P}_l^m(x)$

The associated Legendre polynomials, $P_l^m(x)$, are given by

$$P_l^m(x) = \frac{(-1)^m}{2^l l!}(1-x^2)^{m/2}\frac{d^{l+m}}{dx^{l+m}}(x^2-1)^l \tag{12.19}$$

for integers l and m such that $l \geq 0$ and $-l \leq m \leq l$. The Condon-Shortley phase is included in the definition of the associated Legendre polynomials. The following identity relates positive and negative m

$$P_l^{-m}(x) = (-1)^m \frac{(l-m)!}{(l+m)!}P_l^m(x) \tag{12.20}$$

The normalized associated Legendre polynomials $\widetilde{P}_l^m(x)$ are recommended for scattering computations because they avoid direct computation of factorials. These are given by

$$\widetilde{P}_l^m(x) = \sqrt{\frac{(l+\frac{1}{2})(l-m)!}{(l+m)!}}P_l^m(x) \tag{12.21}$$

where

$$\widetilde{P}_l^{-m}(x) = (-1)^m \widetilde{P}_l^m(x) \tag{12.22}$$

The routine `Plm` returns the normalized associated Legendre polynomials for $l = 0, ..., L$, all $\pm m$, linearly indexed (see Chapter 3 on linear indexing). The result is returned with harmonics indexed along rows, and evaluation points along columns. It calls Matlab's `legendre` with the `'norm'` option, which returns $m = 0, ..., l$ and an extra factor of $(-1)^m$. `Plm` takes out the extra $(-1)^m$ from Matlab's normalization so that the one factor of $(-1)^m$ in (12.19) remains.

```
function [plm] = Plm(L,x)
% Normalized associated Legendre polynomials evaluated at
% l = 0,...,L, m = -l:l.
%
%  \tildeP_l^m(x) = (-1)^m sqrt((l+1/2)(l-m)!/(l+m)!) P_l^m(x)
%
% L:    maximum degree L
% x:    input arguments, x = [-1 1];
%
% plm:  Normalized associated Legendre polynomial
%
% dependencies: lm2ind

x=x(:);
N = length(x);
tot = L^2 + 2*L + 1;
plm = zeros(tot,N);
for l=0:L,
    indlm0 = l^2 + l + 1;   % lm2ind(l,0,'mono')
    indll = l^2 + 2*l + 1;  % lm2ind(l,l,'mono')
    ind = indlm0:indll;
    plm(ind,:) = legendre(l,x,'norm');
end
for l=1:L,
for m=1:l,
   indlm = l^2 + l + m + 1;  % lm2ind(l,m,'mono')
   indlmm = l^2 + l - m + 1; % lm2ind(l,-m,'mono')
   plm(indlmm,:) = (-1)^m*plm(indlm,:);
end
end
for l=1:L,
for m=-l:l,
   indlm = l^2 + l + m + 1;  % lm2ind(l,m,'mono')
   plm(indlm,:) = (-1)^m*plm(indlm,:);
end
end
```

12.3.4 Normalized Associated Legendre Polynomials Derivative, $d/dx\widetilde{P}_l^m(x)$

The derivative of the normalized associated Legendre polynomials is found by substituting (12.21) into the following non-normalized recurrence relation for the associated Legendre polynomial derivative:

$$(x^2-1)\frac{d}{dx}P_l^m(x) = lxP_l^m(x) - (l+m)P_{l-1}^m(x) \tag{12.23}$$

which gives

$$\frac{d}{dx}\widetilde{P}_l^m(x) = \frac{1}{x^2-1}\left(lx\widetilde{P}_l^m(x) - \sqrt{\frac{(l+1/2)}{(l-1/2)}}\sqrt{(l+m)(l-m)}\widetilde{P}_{l-1}^m(x)\right) \tag{12.24}$$

This recurrence only requires $\widetilde{P}_l^m(x)$ to be computed at degrees less than or equal to l. This is computed by the routine `Plmp`, and has the same structure as `Plm`, which returns $l = 0, ..., L$, all $\pm m$, linearly indexed (see Chapter 3). This version is not suitable at the end points $x = [-1, 1]$. In fact, the difficultly of computing the derivative of the associated Legendre polynomials at the end-points is well-known, and other routines can be found elsewhere. However, we use this routine in scattering problems when the poles do not need to be sampled, for example, when using Gauss-Legendre quadrature as the basis for spherical harmonic transforms (see Chapter 8).

```
function [plmp] = Plmp(L,x)
% Normalized associated Legendre polynomials derivative
% l = 0,...,L, m = -l:l.
%
%  d/dx \tildeP_l^m(x) = 1/(x^2-1)*(lx\tildeP_l^m(x) - ...
%               sqrt((l+1/2)/(l-1/2)*(l+m)(l-m))\tildeP_{l-1}^m(x)
%
% L:    maximum degree L
% x:    input arguments, x = [-1 1];
% str:  'scalar' to include (l,m) = (0,0)
%
% plmp:  Normalized associated Legendre polynomial derivative
%
% dependencies: Plm, lm2ind

x=x(:)';
plm = Plm(L,x);
plmp = zeros(size(plm));
for l = 0:L,
for m = -l:l,
    ind = lm2ind(l,m,'mono');
    if l == 0
        c2 = 0;
    elseif abs(m) <= l-1,
        ind2 = lm2ind(l-1,m,'mono');
        c2 = -sqrt(l+1/2)/sqrt(l-1/2)*sqrt((l+m)*(l-m));
        c2 = c2.*plm(ind2,:);
    else
        c2 = 0;
    end
    c1 = l*x.*plm(ind,:);
    plmp(ind,:) = (1./(x.^2-1)).*(c1 + c2);
end
end
```

12.3.5 Normalized Associated Legendre Polynomials, $m\widetilde{P}_l^m(\cos\theta)/\sin\theta$

The variant $m\widetilde{P}_l^m(\cos\theta)/\sin\theta$ is needed for vector wave functions. Numerically, direct division by $\sin\theta$ is a problem at the poles, however, analytically, dividing by $\sin\theta$ is harmless because $P_l^m(\cos\theta) \sim \sin^m\theta$. With the right recurrence relation, it can be computed directly. Start with the following unnormalized recurrence relation

$$\frac{m}{\sin\theta}P_l^m(\cos\theta) = -\frac{1}{2}\left(P_{l-1}^{m+1}(\cos\theta) + (l+m-1)(l+m)P_{l-1}^{m-1}(\cos\theta)\right) \tag{12.25}$$

Then substituting (12.21) and canceling like factorials

$$\begin{aligned}\frac{m}{\sin\theta}\widetilde{P}_l^m(\cos\theta) &= \frac{1}{2}\sqrt{\frac{l-\frac{1}{2}}{l+\frac{1}{2}}}\left(\sqrt{(l-m)(l-m-1)}\widetilde{P}_{l-1}^{m+1}(\cos\theta)\right.\\ &\quad\left.+\sqrt{(l+m)(l+m-1)}\widetilde{P}_{l-1}^{m-1}(\cos\theta)\right)\end{aligned} \tag{12.26}$$

This is computed by the routine `mPlmsin`, and has similar structure to `Plm`. It returns degrees $l = 1, .., L$, all m, linearly indexed along rows with evaluation points along columns. We exclude the monopole because this variant is only used for vector wave functions. It uses the initial condition

$$\frac{(1)}{\sin\theta}\widetilde{P}_1^1(\cos\theta) = \sqrt{\frac{3}{4}} \tag{12.27}$$

Technically this initial condition should have a negative sign. However, we iterate on the output of the Matlab's `legendre`, which has an extra negative sign. Therefore, we adjust the negative signs at the end of the routine so that the outputs will match a direct computation of the quantity $m\widetilde{P}_l^m(\cos\theta)/\sin\theta$ when using `Plm`.

```
function [mPsin] = mPlmsin(L,theta)
% Normalized Legendre polynomial variant, l = 1,...,L
%
% m/sin(theta) \tilde P_l^m(cos(theta)) = 1/2 sqrt((l-1/2)/(l+1/2)) ...
%           (sqrt((l-m)(l-m-1)) \tilde P_{l-1}^{m+1}(cos(theta)) ...
%            + sqrt((l+m)(l+m-1)) \tilde P_{l-1}^{m-1}(cos(theta))
%
% L:        Maximum harmonic order L
% theta:        Spherical angles in radians
%
% mPsin:          Derivative of normalized Legende polynomial evaluated at theta
%               Dimensions: size(theta) x N
%               N = L^2 + 2*L
%
% Dependencies: lm2ind

N = L^2 + 2*L;
theta = theta(:)';
Npoints = length(theta);
P = zeros(N,Npoints);   % P_l^m
mPsin = P;              % m P_l^m/sin(theta)

% load P_l^m for m >= 0
for l=1:L,
    indlm0 = l^2 + l;  % lm2ind(l,0)
    indll = l^2 + 2*l; % lm2ind(l,l)
    ind = indlm0:indll;
    P(ind,:) = legendre(l,cos(theta),'norm');
end

% m P_l^m/sin(theta)
mPsin(lm2ind(1,1),:) = sqrt(3/4)*ones(1,Npoints);  % (l,m) = (1,1)
for l=2:L,
    c1 = 0.5*sqrt((l+0.5)/(l-0.5));
    for m=1:l,
       indlm = l^2 + l + m;             % lm2ind(l,m)
```

```
       indlm1mm1 = (l-1)^2 + l-1 + m-1; % lm2ind(l-1,m-1)
       mPsin(indlm,:) = c1*sqrt((l+m-1)*(l+m))*P(indlm1mm1,:);
       if l-m > 1
          indlm1mp1 = (l-1)^2 + l-1 + m+1; % lm2ind(l-1,m+1)
          mPsin(indlm,:) = mPsin(indlm,:) + c1*sqrt((l-m)*(l-m-1))*P(indlm1mp1,:);
       end
    end
end

% negative m
for l=1:L,
for m=-l:-1,
    indlm = l^2 + l + m;    % lm2ind(l,m)
    indlmm = l^2 + l - m;   % lm2ind(l,-m)
    mPsin(indlm,:) = -(-1)^(m)*mPsin(indlmm,:);
end
end

% extra (-1)^m
for l=1:L,
for m=-l:l,
    indlm = l^2 + l + m;    % lm2ind(l,m)
    mPsin(indlm,:) = (-1)^(m)*mPsin(indlm,:);
end
end
```

12.3.6 Normalized Associated Legendre Polynomials Derivative, $d/d\theta \widetilde{P}_l^m(\cos\theta)$

The variant $d/d\theta \widetilde{P}_l^m(\cos\theta)$ is the derivative of the normalized associated Legendre polynomials with respect to θ when the argument is $\cos\theta$. Start with the recurrence relation for unnormalized Legendre polynomials

$$\begin{aligned} \frac{d}{d\theta} P_l^m(\cos\theta) &= -\sin\theta \frac{d}{d\cos\theta} P_l^m(\cos\theta) && (12.28) \\ &= -\frac{1}{2}\left((l+m)(l-m+1)P_l^{m-1}(\cos\theta) - P_l^{m+1}(\cos\theta)\right) && (12.29) \end{aligned}$$

Substituting the expression for the normalized Legendre polynomials and canceling like factorials

$$\frac{d}{d\theta}\widetilde{P}_l^m(\cos\theta) = \frac{1}{2}\left(\sqrt{(l+m)(l-m+1)}\widetilde{P}_l^{m-1}(\cos\theta) - \sqrt{(l-m)(l+m+1)}\widetilde{P}_l^{m+1}(\cos\theta)\right) \quad (12.30)$$

This recurrence only requires degrees equal to l. This is given by the routine `Plmp2`, and has similar structure to `Plm`. It returns degrees $l = 1, .., L$, all m, linearly indexed in rows and evaluation points along columns. We exclude the monopole because this variant is used for vector wave functions, and because the derivative for $(l, m) = (0, 0)$ is zero. It uses the initial condition

$$\frac{d}{d\theta}\widetilde{P}_l^0(\cos\theta) = -\sqrt{l(l+1)}\widetilde{P}_l^1(\cos\theta) \quad (12.31)$$

```
function [dP] = Plmp2(L,theta)
% Derivative of normalized Legendre polynomial,
% l = 1,...,L
%
% d/dtheta \tilde P_l^m(cos(theta)) = ...
%           1/2(sqrt((l+m)(l-m+1)) \tilde P_l^{m-1}(cos(theta)) ...
%            + sqrt((l-m)(l+m+1)) \tilde P_l^{m+1}(cos(theta))
%
% L:      Maximum harmonic order L
% theta:        Spherical angles in radians
%
% dP:           Derivative of normalized Legende polynomial evaluated at theta
%         Dimensions: size(theta) x N
%             N = L^2 + 2*L
%
```

```
% Dependencies: lm2ind

N = L^2 + 2*L;
theta = theta(:)';
Npoints = length(theta);
P = zeros(N,Npoints);   % P_l^m
dP = P;                 % d/dtheta P_l^m

% load P_l^m for m >= 0
for l=1:L,
    indlm0 = l^2 + l;  % lm2ind(l,0)
    indll = l^2 + 2*l; % lm2ind(l,l)
    ind = indlm0:indll;
    P(ind,:) = legendre(l,cos(theta),'norm');
end

% d/dtheta P_l^m
for l=1:L,
    indlm0 = l^2 + l;       % lm2ind(l,0)
    indlm1 = l^2 + l + 1;   % lm2ind(l,1)
    dP(indlm0,:) = -sqrt(l*(l+1))*P(indlm1,:);
    for m=1:l,
        indlm = l^2 + l + m;  % lm2ind(l,m)
        if m-1 >= 0
            indlmm1 = l^2 + l + m - 1;
            dP(indlm,:) = 0.5*sqrt((l+m)*(l-m+1))*P(indlmm1,:);
        end
        if m+1 <= l
            indlmp1 = l^2 + l + m + 1; % lm2ind(l,m+1)
            dP(indlm,:) = dP(indlm,:) - 0.5*sqrt((l-m)*(l+m+1))*P(indlmp1,:);
        end
    end
end

% negative m
for l=1:L,
for m=-l:-1,
    indlm = l^2 + l + m; % lm2ind(l,m)
    indlmm = l^2 + l -m; % lm2ind(l,-m)
    dP(indlm,:) = (-1)^(m)*dP(indlmm,:);
end
end

% extra (-1)^m
for l=1:L,
for m=-l:l,
    indlm = l^2 + l + m; % lm2ind(l,m)
    dP(indlm,:) = (-1)^(m)*dP(indlm,:);
end
end
```

12.4 Miscellaneous

12.4.1 FFT Frequency

It is surprisingly hard to remember the frequency samples that match the unshifted two-sided discrete Fast Fourier Transform (FFT), for odd or even length signals. Let f_s be the sample frequency and N be the number of points, then the discrete frequencies of the FFT are:

N even:

$$f[k] = \frac{f_s}{N} \begin{cases} k, & k = 0, ..., \dfrac{N}{2} - 1 \\ k - N, & k = \dfrac{N}{2}, ..., N - 1 \end{cases} \tag{12.32}$$

N odd:

$$f[k] = \frac{f_s}{N} \begin{cases} k, & k = 0, ..., \dfrac{N-1}{2} \\ k - N, & k = \dfrac{N+1}{2}, ..., N - 1 \end{cases} \tag{12.33}$$

The routine `fftfreq` takes as input the sampling rate, f_s, and number of points, N, and returns the discrete frequencies $f[k]$ that correspond to the unshifted two-sided FFT of odd or even N.

```
function [f] = fftfreq(fs,N)
% Two-sided FFT frequencies
%
% fs:   sample rate
% N:    number of samples
%
% f:    fft frequencies (same units as fs)

if mod(N,2) == 0    % N even
    seg = 1:(N/2-1);
    f = [0 seg -N/2 -fliplr(seg)]'*fs/N;
else                % N odd
    seg = 1:((N-1)/2);
    f = [0 seg -fliplr(seg)]'*fs/N;
end
```

12.4.2 Phase wrapping

Phase wrapping is easily done with $\angle \exp(i\phi)$, but there is a fun variation that uses nearest integer rounding written in terms of elementary functions:

$$\Phi(\phi) = \phi - 2\pi \left[\frac{\phi}{2\pi} \right] \tag{12.34}$$

$$= -2 \arctan \cot \left(\frac{\phi}{2} + \frac{\pi}{2} \right) \tag{12.35}$$

where $[\cdot]$ means round to the nearest integer so that $-\pi \leq \Phi \leq \pi$.

```
function [Phi] = wrap(phi)
% Phase wrapping with analytical functions
%
% phi:  phase (rad)
%
% Phi:  wrapped phase (rad) on [-pi pi]

    Phi = -2*atan(cot(phi/2 + pi/2));
end
```

12.4.3 Cubic Roots

The roots of a cubic polynomial are known analytically. Let the cubic be written

$$ax^3 + bx^2 + cx + d = 0 \tag{12.36}$$

and define

$$w = -\frac{b}{3a} \tag{12.37}$$

$$Q = \frac{3ac - b^2}{9a^2} \tag{12.38}$$

$$R = \frac{9abc - 27a^2d - 2b^3}{54a^3} \tag{12.39}$$

$$D = Q^3 + R^2 \tag{12.40}$$

When $D \geq 0$, there is one real root and two complex conjugate roots. For this case, define

$$S = \sqrt[3]{R + \sqrt{D}} \tag{12.41}$$

$$T = \sqrt[3]{R - \sqrt{D}} \tag{12.42}$$

When $D < 0$, there are three real roots and the following is used

$$S = \sqrt[3]{\rho}e^{i\theta/3} \tag{12.43}$$

$$T = \sqrt[3]{\rho}e^{-i\theta/3} \tag{12.44}$$

$$\rho = \sqrt{-Q^3} \tag{12.45}$$

$$\theta = \arccos\left(\frac{R}{\rho}\right) \tag{12.46}$$

In all instances, the positive roots of $\sqrt[3]{}$ and $\sqrt{}$ are used. The roots of the cubic are then given by

$$x_1 = w + S + T \tag{12.47}$$

$$x_2 = w - \frac{1}{2}(S + T) + i\frac{\sqrt{3}}{2}(S - T) \tag{12.48}$$

$$x_3 = w - \frac{1}{2}(S + T) - i\frac{\sqrt{3}}{2}(S - T) \tag{12.49}$$

where x_1 is always real.

The routine `cubicroots` is a vectorized implementation of the solution above that computes the roots of multiple cubic polynomials simultaneously. Coefficient arrays can be any size. When `nargout == 1`, the routine returns the guaranteed real root x_1. The key difference between this routine and Matlab's `roots` is that `roots` is not vectorized and only solves one polynomial at a time, otherwise, the two routines produce the same solutions.

```
function [x1 x2 x3] = cubicroots(a,b,c,d)
% Vectorized roots of a cubic polynomial
%
%     ax^3 + bx^2 + cx + d = 0
%
% a,b,c,d       Coefficients, any size
%
% x1,[x2,x3]    Polynomial roots, same size as a,b,c,d
%               If nargout==1, function returns one real root, x1

% Q, R, and D parameters
Q = (3*a.*c-b.^2)./(9*a.^2);
R = (9*a.*b.*c-27*a.^2.*d-2*b.^3)./(54*a.^3);
```

```
D = Q.^3 + R.^2;
w = -b./(3*a);
S = zeros(size(D));
T = zeros(size(D));

% condition for one real root, and two complex
I1 = (D>=0);
S(I1) = nthroot(R(I1)+sqrt(D(I1)),3);
T(I1) = nthroot(R(I1)-sqrt(D(I1)),3);

% condition for three real roots
I2 = ~I1;
rho = nthroot(-Q(I2).^3,2);
theta = acos(R(I2)./rho);
S(I2) = nthroot(rho,3).*exp(1i*theta/3);
T(I2) = nthroot(rho,3).*exp(-1i*theta/3);

% roots
x1 = real(w + S + T);
x2 = w - (1/2)*(S+T) + (1i*sqrt(3)/2)*(S-T);
x3 = w - (1/2)*(S+T) - (1i*sqrt(3)/2)*(S-T);
```

12.4.4 Quartic Roots

Like the cubic, the roots of a quartic polynomial are known analytically, [80]. Let the quartic be written

$$ax^4 + bx^3 + cx^2 + dx + e = 0 \tag{12.50}$$

This is put in standard form by dividing the leading coefficient

$$x^4 + a_3x^3 + a_2x^2 + a_1x + a_0 = 0 \tag{12.51}$$

where $a_3 = b/a$, $a_2 = c/a$, $a_1 = d/a$ and $a_0 = e/a$. This has resolvent cubic

$$y^3 - a_2y^2 + (a_1a_3 - 4a_0)y + (4a_2a_0 - a_1^2 - a_3^2a_0) = 0 \tag{12.52}$$

If y_1 is any real root of (12.52) (there is always at least one), then the roots of the quartic are given by

$$x_{1,2} = w + \frac{1}{2}R \pm \frac{1}{2}D \tag{12.53}$$

$$x_{3,4} = w - \frac{1}{2}R \pm \frac{1}{2}E \tag{12.54}$$

where

$$D = \begin{cases} \sqrt{T - R^2 + \dfrac{U}{R}}, & R \neq 0 \\ \sqrt{T + V}, & R = 0 \end{cases} \tag{12.55}$$

$$E = \begin{cases} \sqrt{T - R^2 - \dfrac{U}{R}}, & R \neq 0 \\ \sqrt{T - V}, & R = 0 \end{cases} \tag{12.56}$$

and

$$w = -\frac{1}{4}a_3 \tag{12.57}$$

$$R = \sqrt{\frac{a_3^2}{4} - a_2 + y_1} \tag{12.58}$$

$$T = \frac{3a_3^2}{4} - 2a_2 \tag{12.59}$$

$$U = \frac{4a_3a_2 - 8a_1 - a_3^3}{4} \tag{12.60}$$

$$V = 2\sqrt{y_1^2 - 4a_0} \tag{12.61}$$

The routine `quarticroots` is a vectorized implementation of the solution above that computes the roots of multiple quartic polynomials simultaneously. Coefficient arrays can be any size. The real cubic root, y_1, comes from `cubicroots`. Again, Matlab's `roots` only solves one polynomial at a time, otherwise the two routines produce the same solutions.

```
function [x1 x2 x3 x4] = quarticroots(a,b,c,d,e)
% Vectorized roots of a quartic polynomial
%
%     ax^4 + bx^3 + cx^2 + dx + e = 0
%
% a,b,c,d,c     Coefficients, any size
%
% x1,x2,x3,x4   Polynomial roots, same size as a,b,c,d,e
%
% Dependencies: cubicroots

% put quartic in standard form
a3 = b./a;
a2 = c./a;
a1 = d./a;
a0 = e./a;

% coefficients of the resolvent cubic
cube_a = ones(size(a3));
cube_b = -a2;
cube_c = a1.*a3 - 4*a0;
cube_d = 4*a2.*a0 - a1.^2 - a3.^2.*a0;

% one real root of the cubic
y1 = cubicroots(cube_a,cube_b,cube_c,cube_d);

% compute the parameters
R = sqrt((1/4)*a3.^2 - a2 + y1);
w = -(1/4)*a3;
T = (3/4)*a3.^2 - 2*a2;
U = (1/4)*(4*a3.*a2-8*a1-a3.^3)./R;
V = 2*sqrt(y1.^2-4*a0);

D = zeros(size(a3));
E = zeros(size(a3));
% condition for R ~= 0
I1 = (R~=0);
D(I1) = sqrt(T(I1)-R(I1).^2+U(I1));
E(I1) = sqrt(T(I1)-R(I1).^2-U(I1));

% condition for R = 0
I2 = ~I1;
D(I2) = sqrt(T(I2)+V(I2));
E(I2) = sqrt(T(I2)-V(I2));

% four roots
x1 = w+R/2+D/2;
x2 = w+R/2-D/2;
x3 = w-R/2+E/2;
x4 = w-R/2-E/2;
```

12.4.5 Uniform Points on a Sphere

A quick approximation of uniform points on a sphere is based on the disco ball. Points are distributed mostly evenly at discrete latitudes centered on a prime meridian. This avoids crowding at the poles that happens with uniform sampling in spherical coordinates. Specifically, the number of lines of latitude is chosen which are divided equally between $\theta = [0, \pi]$, then each circle of constant latitude is divided by the latitude spacing as many times as it will go.

The routine `discoball` takes as input the number of latitude lines `nlat` and returns the spherical coordinates of the points (θ, ϕ).

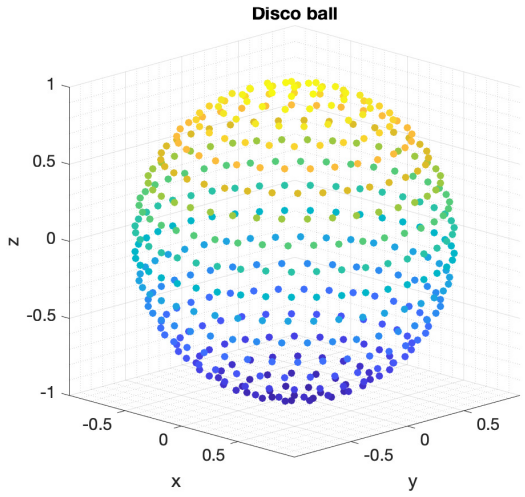


Figure 12.1: Disco ball approximation for uniform points on a sphere

```
function [th, phi] = discoball(nlat)
% Approximate uniform distribution of points on a sphere
% using a disco ball method.
%
% nlat:         number of latitude lines
%
% th, phi:      spherical angles of points, in radians.

th = linspace(0,pi,nlat)';
dth = th(2)-th(1);
xx = [];
for n=1:nlat,
    rad = sin(th(n));
    nlon = round(2*pi*rad/dth)+1;
    dlon = 2*pi/nlon;
    lon = linspace(0,2*pi-dlon,nlon)';
    xx = [xx; th(n)*ones(nlon,1) lon];
end
th = xx(:,1);
phi = xx(:,2);
```

12.4.6 Uniformly Random Points on a Sphere

To create a set of points that are uniformly random over the unit sphere, it is not correct to draw from a uniformly random distribution of spherical angles (θ, ϕ), because this leads to crowding at the poles. Instead, the following transformation is used, [81],

$$\phi = 2\pi U(0,1) \tag{12.62}$$
$$\theta = \arccos(2V(0,1) - 1) \tag{12.63}$$

where $U(0,1)$ and $V(0,1)$ are uniform random variables from $[0,1]$.

The routine `randsphere` takes as input the total number of points, N, and returns the spherical (θ, ϕ) coordinates of the points.

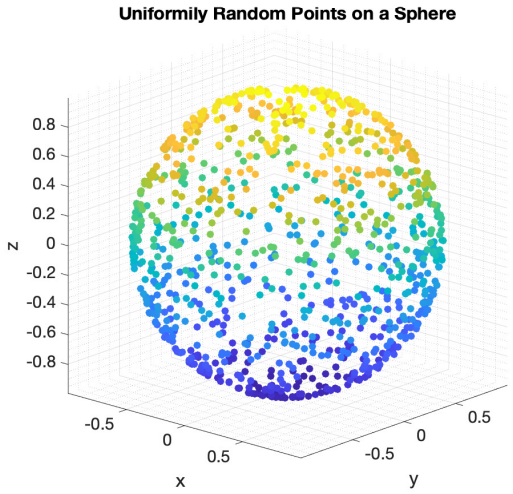


Figure 12.2: Uniformly random points on a sphere, $N = 1000$.

```
function [th, phi] = randsphere(N)
% Uniformly random points on a sphere
%
% N:        number of points
%
% th, phi:  spherical angles of points, in radians.

th = acos(2*rand(N,1) - 1);
phi = 2*pi*rand(N,1);
```

Bibliography

[1] C.-T. Tai, *Dyadic Green Functions in Electromagnetic Theory*. Institute of Electrical & Electronics Engineers (IEEE), 1994.

[2] W. C. Chew, *Waves and Fields in Inhomogeneous Media*, vol. 522. New York: IEEE Press, 1995.

[3] L. Tsang, J. A. Kong, and R. T. Shin, *Theory of Microwave Remote Sensing*. New York: Wiley Interscience, 1985.

[4] A. Devaney, "Geophysical diffraction tomography," *IEEE Transactions on Geoscience and Remote Sensing*, no. 1, pp. 3–13, 1984.

[5] G. Gao, C. Torres-Verdin, and T. M. Habashy, "Analytical techniques to evaluate the integrals of 3D and 2D spatial dyadic Green's functions," *Progress In Electromagnetics Research*, vol. 52, pp. 47–80, 2005.

[6] P. M. Van Den Berg and R. E. Kleinman, "A contrast source inversion method," *Inverse problems*, vol. 13, no. 6, p. 1607, 1997.

[7] A. F. Peterson, S. L. Ray, R. Mittra, I. of Electrical, and E. Engineers, *Computational Methods for Electromagnetics*, vol. 2. IEEE press New York, 1998.

[8] M. Haynes and M. Moghaddam, "Vector Green's function for S-parameter measurements of the electromagnetic volume integral equation," *IEEE Transactions on Antennas and Propagation*, vol. 60, no. 3, pp. 1400–1413, 2012.

[9] T. J. Cui, W. C. Chew, X. X. Yin, and W. Hong, "Study of resolution and super resolution in electromagnetic imaging for half-space problems," *IEEE Transactions on Antennas and Propagation*, vol. 52, no. 6, pp. 1398–1411, 2004.

[10] L. Tsang, J. Kong, and K. Ding, *Scattering of Electromagnetic Waves, Vol. 1: Theory and Applications*. New York: Wiley Interscience, 2000.

[11] R. Fitzpatrick, "Angular momentum operators." http://farside.ph.utexas.edu/teaching/jk1/lectures/node123.html. Last accessed: 2020-06-29.

[12] J. D. Jackson, *Classical Electrodynamics*. American Association of Physics Teachers, 1999.

[13] A. D. Yaghjian, "Sampling criteria for resonant antennas and scatterers," *Journal of applied physics*, vol. 79, no. 10, pp. 7474–7482, 1996.

[14] A. R. Edmonds, *Angular Momentum in Quantum Mechanics*. Princeton University Press, 1996.

[15] J. E. Hansen, *Spherical Near-Field Antenna Measurements*, vol. 26. Iet, 1988.

[16] S. Stein, "Addition theorems for spherical wave functions," *Quart. Appl. Math.*, vol. 19, 1961.

[17] T. J. Dufva, J. Sarvas, and J. C.-E. Sten, "Unified derivation of the translational addition theorems for the spherical scalar and vector wave functions," *Progress In Electromagnetics Research B*, vol. 4, pp. 79–99, 2008.

[18] E. Wigner, *Group Theory: and its Application to the Quantum Mechanics of Atomic Spectra*, vol. 5. Elsevier, 2012.

[19] "Wigner d-matrix." https://en.wikipedia.org/wiki/Wigner_D-matrix. Last accessed: 2021-01-16.

[20] C. H. Choi, J. Ivanic, M. S. Gordon, and K. Ruedenberg, "Rapid and stable determination of rotation matrices between spherical harmonics by direct recursion," *The Journal of Chemical Physics*, vol. 111, no. 19, p. 8825, 1999.

[21] Z. Gimbutas and L. Greengard, "A fast and stable method for rotating spherical harmonic expansions," *Journal of Computational Physics*, vol. 228, no. 16, pp. 5621–5627, 2009.

[22] D. W. Mackowski, "Analysis of radiative scattering for multiple sphere configurations," *Proceedings of the Royal Society of London. Series A: Mathematical and Physical Sciences*, vol. 433, no. 1889, pp. 599–614, 1991.

[23] W. C. Chew, "Recurrence relations for three-dimensional scalar addition theorem," *Journal of Electromagnetic Waves and Applications*, vol. 6, no. 1-4, pp. 133–142, 1992.

[24] W. C. Chew and Y. Wang, "Efficient ways to compute the vector addition theorem," *Journal of electromagnetic waves and applications*, vol. 7, no. 5, pp. 651–665, 1993.

[25] X. Duan, M. Haynes, and M. Moghaddam, "Experimental verification of the recursive T-matrix method solutions at microwave frequencies," *IEEE Transactions on Antennas and Propagation*, vol. 63, no. 12, pp. 5727–5740, 2015.

[26] L. Zhang, "Generalized optical theorem for an arbitrary incident field," *The Journal of the Acoustical Society of America*, vol. 145, no. 3, pp. EL185–EL189, 2019.

[27] M. A. Yurkin and A. G. Hoekstra, "The discrete dipole approximation: an overview and recent developments," *Journal of Quantitative Spectroscopy and Radiative Transfer*, vol. 106, no. 1-3, pp. 558–589, 2007.

[28] J. J. van Zyl, *Synthetic Aperture Radar Polarimetry*, vol. 2. John Wiley & Sons, 2011.

[29] K. Sarabandi and T. Senior, "Low-frequency scattering from cylindrical structures at oblique incidence," *IEEE transactions on geoscience and remote sensing*, vol. 28, no. 5, pp. 879–885, 1990.

[30] J. M. Stiles and K. Sarabandi, "A scattering model for thin dielectric cylinders of arbitrary cross section and electrical length," *IEEE Transactions on Antennas and Propagation*, vol. 44, no. 2, pp. 260–266, 1996.

[31] R. Schiffer and K. Thielheim, "Light scattering by dielectric needles and disks," *Journal of Applied Physics*, vol. 50, no. 4, pp. 2476–2483, 1979.

[32] P. Waterman, "Matrix formulation of electromagnetic scattering," *Proceedings of the IEEE*, vol. 53, no. 8, pp. 805–812, 1965.

[33] C.-T. Tai, *General Vector and Dyadic Analysis: Applied Mathematics in Field Theory*, vol. 9. Wiley-IEEE Press, 1997.

[34] L. Greengard and V. Rokhlin, "A fast algorithm for particle simulations," *Journal of computational physics*, vol. 135, no. 2, pp. 280–292, 1997.

[35] J. M. Song and W. C. Chew, "Multilevel fast-multipole algorithm for solving combined field integral equations of electromagnetic scattering," *Microwave and Optical Technology Letters*, vol. 10, no. 1, pp. 14–19, 1995.

[36] A. C. Yucel, "Helmholtz and high-frequency maxwell multilevel fast multipole algorithms with self-tuning library," Master's thesis, 2008.

[37] T. B. Hansen, "Exact plane-wave expansion with directional spectrum: Application to transmitting and receiving antennas," *IEEE transactions on antennas and propagation*, vol. 62, no. 8, pp. 4187–4198, 2014.

[38] J. Song, C.-C. Lu, and W. C. Chew, "Multilevel fast multipole algorithm for electromagnetic scattering by large complex objects," *IEEE Transactions on Antennas and Propagation*, vol. 45, no. 10, pp. 1488–1493, 1997.

[39] J. Song and W. C. Chew, "Error analysis for the truncation of multipole expansion of vector green's functions," *IEEE microwave and wireless components letters*, vol. 11, no. 7, pp. 311–313, 2001.

[40] E. Darve, "The fast multipole method: numerical implementation," *Journal of Computational Physics*, vol. 160, no. 1, pp. 195–240, 2000.

[41] C. H. Beentjes, "Quadrature on a spherical surface," *Working note available on the website http://people.maths.ox.ac.uk/beentjes/Essays*, 2015.

[42] K. Atkinson and W. Han, *Spherical Harmonics and Approximations on the Unit Sphere: An Introduction*, vol. 2044. Springer Science & Business Media, 2012.

[43] O. M. Bucci, C. Gennarelli, and C. Savarese, "Optimal interpolation of radiated fields over a sphere," *IEEE Transactions on Antennas and Propagation*, vol. 39, no. 11, pp. 1633–1643, 1991.

[44] R. Jakob-Chien and B. K. Alpert, "A fast spherical filter with uniform resolution," *Journal of Computational Physics*, vol. 136, no. 2, pp. 580–584, 1997.

[45] B. Shanker, A. A. Ergin, M. Lu, and E. Michielssen, "Fast analysis of transient electromagnetic scattering phenomena using the multilevel plane wave time domain algorithm," *IEEE Transactions on Antennas and Propagation*, vol. 51, no. 3, pp. 628–641, 2003.

[46] A. Dutt, M. Gu, and V. Rokhlin, "Fast algorithms for polynomial interpolation, integration, and differentiation," *SIAM Journal on Numerical Analysis*, vol. 33, no. 5, pp. 1689–1711, 1996.

[47] J. A. Kong, *Electromagnetic Wave Theory*. Wiley, 1986.

[48] F. T. Ulaby, *Fundamentals of Applied Electromagnetics*. Upper Saddle River, NJ: Prentice-Hall, 1999.

[49] K. Trott, "The disk: A comparison of electromagnetic scattering solutions and its use as a calibration standard for bistatic RCS (radar cross section) measurements (report for sep. 1986- sep. 1987)," 1988.

[50] "Double integral over an arbitrary triangle." https://math.stackexchange.com/questions/954409/double-integral-over-an-arbitrary-triangle. Last accessed: 2020-07-22.

[51] C. A. Mack, "Analytic form for the power spectral density in one, two, and three dimensions," *Journal of Micro/Nanolithography, MEMS, and MOEMS*, vol. 10, no. 4, p. 040501, 2011.

[52] C. A. Mack, "Generating random rough edges, surfaces, and volumes," *Applied Optics*, vol. 52, no. 7, pp. 1472–1480, 2013.

[53] L. Tsang, J. Kong, K. Ding, and C. Ao, *Scattering of Electromagnetic Waves, Vol. 2: Numerical Simulations*. New York: Wiley Interscience, 2001.

[54] M. K. Shepard, R. A. Brackett, and R. E. Arvidson, "Self-affine (fractal) topography: Surface parameterization and radar scattering," *Journal of Geophysical Research: Planets*, vol. 100, no. E6, pp. 11709–11718, 1995.

[55] M. K. Shepard and B. A. Campbell, "Radar scattering from a self-affine fractal surface: Near-nadir regime," *Icarus*, vol. 141, no. 1, pp. 156–171, 1999.

[56] D. P. Kroese and Z. I. Botev, "Spatial process simulation," in *Stochastic geometry, spatial statistics and random fields*, pp. 369–404, Springer, 2015.

[57] K.-H. Ding, X. Xu, and L. Tsang, "Electromagnetic scattering by bicontinuous random microstructures with discrete permittivities," *IEEE transactions on geoscience and remote sensing*, vol. 48, no. 8, pp. 3139–3151, 2010.

[58] X. Xu, L. Tsang, and S. Yueh, "Electromagnetic models of co/cross polarization of bicontinuous/DMRT in radar remote sensing of terrestrial snow at X-and Ku-band for CoReH2O and SCLP applications," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 5, no. 3, pp. 1024–1032, 2012.

[59] W. Chang, K.-H. Ding, L. Tsang, and X. Xu, "Microwave scattering and medium characterization for terrestrial snow with QCA-Mie and bicontinuous models: comparison studies," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 54, no. 6, pp. 3637–3648, 2016.

[60] S. Tan, C. Xiong, X. Xu, and L. Tsang, "Uniaxial effective permittivity of anisotropic bicontinuous random media using NMM3D," *IEEE Geoscience and Remote Sensing Letters*, vol. 13, no. 8, pp. 1168–1172, 2016.

[61] J. H. G. Alves, M. L. Banner, and I. R. Young, "Revisiting the Pierson-Moskowitz asymptotic limits for fully developed wind waves," *Journal of physical oceanography*, vol. 33, no. 7, pp. 1301–1323, 2003.

[62] H. L. Tolman, "User manual and system documentation of WAVEWATCH III TM version 3.14," *Technical note, MMAB Contribution*, no. 276, 2009.

[63] W. J. Plant, "The ocean wave height variance spectrum: wavenumber peak versus frequency peak," *Journal of Physical Oceanography*, vol. 39, no. 9, pp. 2382–2383, 2009.

[64] G. Soriano, M. Joelson, and M. Saillard, "Doppler spectra from a two-dimensional ocean surface at L-band," *Geoscience and Remote Sensing, IEEE Transactions on*, vol. 44, no. 9, pp. 2430–2437, 2006.

[65] D. Hasselmann, M. Dunckel, and J. Ewing, "Directional wave spectra observed during JONSWAP 1973," *Journal of Physical Oceanography*, vol. 10, no. 8, pp. 1264–1280, 1980.

[66] J. Niedzwecki and C. Whatley, "A comparative study of some directional sea models," *Ocean Engineering*, vol. 18, no. 1, pp. 111–128, 1991.

[67] A. Osborne, *Nonlinear Ocean Waves & the Inverse Scattering Transform*, vol. 97. Access Online via Elsevier, 2010.

[68] K. Muinonen, T. Nousiainen, P. Fast, K. Lumme, and J. Peltoniemi, "Light scattering by gaussian random particles: ray optics approximation," *Journal of Quantitative Spectroscopy and Radiative Transfer*, vol. 55, no. 5, pp. 577–601, 1996.

[69] A. A. R. Neves, L. A. Padilha, A. Fontes, E. Rodriguez, C. d. B. Cruz, L. C. Barbosa, and C. L. Cesar, "Analytical results for a Bessel function times Legendre polynomials class integrals," *Journal of Physics A: Mathematical and General*, vol. 39, no. 18, p. L293, 2006.

[70] G. Gouesbet and J. A. Lock, "Rigorous justification of the localized approximation to the beam-shape coefficients in generalized Lorenz–Mie theory. II. Off-axis beams," *JOSA A*, vol. 11, no. 9, pp. 2516–2525, 1994.

[71] "*NIST Digital Library of Mathematical Functions*." http://dlmf.nist.gov/, Release 1.0.26 of 2020-03-15. F. W. J. Olver, A. B. Olde Daalhuis, D. W. Lozier, B. I. Schneider, R. F. Boisvert, C. W. Clark, B. R. Miller, B. V. Saunders, H. S. Cohl, and M. A. McClain, eds.

[72] J. Liu, Y. Kuga, A. Ishimaru, X. Pi, and A. Freeman, "Ionospheric effects on SAR imaging: a numerical study," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 41, no. 5, pp. 939–947, 2003.

[73] A. Ishimaru, Y. Kuga, J. Liu, Y. Kim, and T. Freeman, "Ionospheric effects on synthetic aperture radar at 100 MHz to 2 GHz," *Radio Science*, vol. 34, no. 1, pp. 257–268, 1999.

[74] N. C. Rogers, S. Quegan, J. S. Kim, and K. P. Papathanassiou, "Impacts of ionospheric scintillation on the BIOMASS P-band satellite SAR," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 52, no. 3, pp. 1856–1868, 2013.

[75] A. Ishimaru, *Wave Propagation and Scattering in Random Media*, vol. 2. Academic press New York, 1978.

[76] D. Eberly, "Computing a point of reflection on a sphere." https://www.geometrictools.com/. March 2, 2008.

[77] "Line-sphere intersection." https://en.wikipedia.org/wiki/Line\OT1\textendashsphere_intersection. Last accessed: 2021-06-11.

[78] F. Hélière, C.-C. Lin, H. Corr, and D. Vaughan, "Radio echo sounding of Pine Island Glacier, West Antarctica: aperture synthesis processing and analysis of feasibility from space," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 45, no. 8, pp. 2573–2582, 2007.

[79] Y. Lei, M. S. Haynes, D. Arumugam, and C. Elachi, "A 2D pseudospectral time-domain (PSTD) simulator for large-scale electromagnetic scattering and radar sounding applications," *IEEE Transactions on Geoscience and Remote Sensing*, 2020.

[80] E. W. Weisstein, "Quartic Equation." From MathWorld—A Wolfram Web Resource, https://mathworld.wolfram.com/QuarticEquation.html. Last visited on May 6, 2020.

[81] E. W. Weisstein, "Sphere Point Picking." From MathWorld—A Wolfram Web Resource, https://mathworld.wolfram.com/SpherePointPicking.html. Last accessed: 2021-4-7.