The Dissertation Committee for Kort Alan Travis
certifies that this is the approved version of the following dissertation:

Optical Scattering from Nanoparticle Aggregates

Committee:

Ernst-Ludwig Florin, Supervisor

Konstantin V. Sokolov, Supervisor

Michael C. Downer

Michael Marder

Gennady Shvets

Optical Scattering from Nanoparticle Aggregates

by

Kort Alan Travis, B.S.; M.A.

DISSERTATION

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

DOCTOR OF PHILOSOPHY

THE UNIVERSITY OF TEXAS AT AUSTIN

December 2010

Acknowledgments

My supervisors and committee, for guidance and patience:

- Konstantin Sokolov and Ernst-Ludwig Florin, at the University of Texas at Austin;
- Mike Downer, Mike Marder, and Gennady Shvets, at the University of Texas at Austin;
- Jochen Guck at the University of Leipzig.

My mentors and professors, for excellent teaching and inspiration:

- Marshall F. Onellion, Kirk W. McVoy, and Edward E. Miller, at the University of Wisconsin at Madison;
- John Wilkerson, Marjorie A. Olmstead, Frederick C. Brown, and Hong Qian, at the University of Washington at Seattle;
- Vadim Kaplunovski, Takeshi Udagawa, and Thomas A. Griffy, at the University of Texas at Austin.

My research colleagues and fellow students, for assistance and friendship:

- Zhengui Wang at Seattle;
- Timo Betz, Bryan Lincoln, Florian Rückerl and Dorothea Schön at Leipzig;

- Kelly Rodibaugh, Dmitry Culcur, Wendy Chih-Wen Kan, Ryna Kamik, Tim Larson, and Justina Tam at Austin.

My parents and siblings, for encouragement and love.

And the many others who have enriched and elevated my life and education in innumerable ways.

Optical Scattering from Nanoparticle Aggregates

Kort Alan Travis, Ph.D.
The University of Texas at Austin, 2010

Supervisors: Ernst-Ludwig Florin
Konstantin V. Sokolov

Nanometer-scale particles of the noble metals have been used for decades as contrast enhancement agents in electron microscopy. Over the past several years it has been demonstrated that these particles also function as excellent contrast agents for optical imaging techniques. The resonant optical scattering they exhibit enables scattering cross sections that may be many orders of magnitude greater than the analogous efficiency factor for fluorescent dye molecules. Biologically relevant labeling with nanoparticles generally results in aggregates containing a few to several tens of particles. The electrodynamic coupling between particles in these aggregates produces observable shifts in the resonance-scattering spectrum. This dissertation provides a theoretical analysis of the scattering from nanoparticle aggregates. The key objectives are to describe this scattering behavior qualitatively and to provide numerical codes usable for modeling its application to biomedical engineering. Considerations of the lowest-order dipole-dipole coupling lead to simple qualitative predictions of the behavior of the spectral properties of the optical cross sections as they depend on number of particles, inter-particle spacing, and aggregate aspect ratio. More comprehensive analysis using the multiple-particle T-matrix formalism allows the elaboration of more subtle cross-section spectral features depending on the interactions of the electrodynamic collective-modes of the

aggregate, of individual-particle modes, and of modes associated with groups of particles within the aggregate sub-structure. In combination these analyses and the supporting numerical code-base provide a unified electrodynamic approach which facilitates interpretation of experimental cross section spectra, guides the design of new biophysical experiments using nanoparticle aggregates, and enables optimal fabrication of nanoparticle structures for biophysical applications.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

For centuries, nanometer-scale particles of the noble metals have been used for the coloring of glass [1]. For example, if gold is dissolved in the hot glass melt and appropriate thermal annealing is applied after solidification, a range of colors from deep ruby red to violet can be produced. These colors are associated with the extinction cross section of a dispersion of gold particles in the glass with mean diameter of about $40\mathrm{nm}$. Colloidal dispersions of metals in an appropriate solvent manifest similar effects, with the specifics of the transmitted and reflected colors depending critically on the complex permittivity of the metal, and somewhat less critically on particle size [2]. The first relatively complete theory explaining this behavior was developed by Gustav Mie and published in 1908 [3]. In 1909, Peter Debye extended Mie's theory and provided additional mathematical interpretation [4]. A hundred years later, it is still largely Debye's work that is used when drawing qualitative generalizations from Mie scattering theory.

During the late 1970s and the 1980s, observations stimulated by the popularity of surface enhanced Raman scattering (SERS) led to recognition of the significant enhancement of the electric field at the surface of particles in an optical field [5, 6]. An implication of this field enhancement is the dipole-dipole coupling between closely spaced particles, which produces a wavelength shift and a broadening in the scattering and extinction cross sections [7].

The labeling of molecules of biological interest with antibody-conjugated colloidal

gold (Au) is the so-called *immunogold* technique. Over the last several decades this technique has been centrally important in enabling the use in biological and medical research of transmission electron microscopy (TEM)[8, 9, 10] and, to a lesser degree, of scanning electron microscopy (SEM)[11, 12, 13]. The high electron density of Au (for TEM) and its efficient emission of secondary and back-scattered electrons (for SEM) are associated with significantly enhanced contrast compared to unlabeled biological materials.

Over the past several years it has been demonstrated that colloidal particles function as excellent contrast enhancement agents for optical imaging techniques as well. As might be expected, the initial use of nanoparticles in optical techniques started with SERS [14, 15, 16, 17, 18] but was quickly expanded to include more common imaging techniques [19, 20, 21, 22]. The resonant optical scattering exhibited by these particles enables scattering cross sections that may be many orders of magnitude greater than the analogous efficiency factor for fluorescent dye molecules. In addition, the conjugation chemistry developed and validated for electron microscopy can be used essentially without modification.

For diagnostic applications, especially in molecular cell biology, the target molecules often occur in dimers, or in clusters containing a few tens of molecules. A result is that the antibody-conjugated nanoparticles bind as pairs, or as aggregates with number distribution analogous to that of the target molecules [22]. In many practical cases the spacing between nanoparticles is small enough that the resulting dipole-dipole coupling produces significant changes in the scattering spectrum [7, 19, 22].

Although essentially complete in its physical formulation, Mie's analysis can be generalized and expressed in a more convenient modern form, in particular, in the form stipulated by T-matrix (transfer-operator) theory, which was developed by Wa-

terman and published in 1965[23]. The articles of Barber and Yeh [24], Kiselev et al. [25], and Mishchenko et al. [26] exemplify state of the art for application of the T-matrix formalism. These articles present an approach that is practical for the treatment of arbitrary incident fields and arbitrarily shaped (albeit homogeneous) dielectric bodies. The extension of this approach to dielectric bodies with volume inhomogeneity appears feasible. The article by Barber and Yeh is especially valuable because it provides a qualitative comparison of scattering from spherical and non-spherical bodies, in addition to providing a useful over-all summary of work in the field regarding non-spherical bodies. Although the approach discussed in all three of these articles is presented in terms of scattering from dielectric bodies, it is applicable with minimal modification to scattering from metals.

Gouesbet et al. [27, 28] present a much less useful "generalized Mie technique." These authors deal with generalized incident fields for a spherical dielectric body but, unfortunately, their method of generalization does not easily lend itself to qualitative interpretation. To their credit, however, their results do have some practical application given their treatment of a Gaussian incident field, and they do provide useful results for radiative balance and radiation pressure effects.

Much modern theoretical work dealing with electrodynamic scattering focuses on scattering from aggregates composed of multiple scattering centers. This work includes several specialties such as the theory of the *effective* dielectric constant of composite nanoparticle media [29, 30], the modeling of photonic crystal structures [31], and the scattering theory of terrestrial clouds and astrophysical nebulae [26, 32, 33].

In principle, once the exact co-ordinates of the particles making up an aggregate are known, the scattering cross section can be calculated exactly. For the case of an aggregate composed of multiple, independently scattering particles, this calcu-

lation is accomplished by averaging the single particle cross section over a suitable statistical distribution [26]. When scattering from the particles is not independent, although the theory is much more complicated to implement, it is well understood [34, 35]. In practice, the calculation involved is tractable only for aggregates comprised of a limited number of particles and, even if the scattering cross section can be calculated exactly or arrived at using numerical techniques, it is difficult to infer qualitative behavior.

A successful approach to obtaining qualitative behavior for scattering from aggregates has been to focus on the transition from a collection of independently scattering objects to a collection of objects undergoing dipole-dipole coupling (or coupling of higher multipole order). With this approach it is possible to understand the qualitative effects of coupling on the scattering cross section.

With respect to qualitative behavior, it is also useful to focus on scattering from random aggregates. It should not be surprising that, given his substantial contributions to Mie theory, Debye initiated work in this area [36, 37]. As with many sub-fields of physics, for its most general case this is an area awaiting development of suitable mathematical theory [32]. However, when the effect of random particle positions can be quantified, it is often discovered that studying the scattering from an ideal non-random aggregate provides quite useful insight into the scattering behavior of the actual structure [38, 39, 40, 41, 42]. In many of the applications of interest to this dissertation, the properties of the ideal aggregate's scattering cross section can be shown to be so close to those of the actual aggregate that they can be treated as identical.

This dissertation presents a theoretical analysis of the scattering from nanoparticle aggregates. The key objectives are to describe the scattering behavior qualitatively and to provide numerical codes usable for modeling applications in biomed-

ical engineering. Thus, although a general analysis is provided where possible, the specifics of possible biomedical applications guide the discussion.

Chapter 2 begins with a discussion of basic electrodynamic considerations, including the definition of a material permittivity function suitable for use with the noble metals. This function is required for the calculation of any cross section, and is experimentally derived except that small phenomenological corrections are made for particle size. The transfer operator (T-matrix) formalism is then presented in the context of the Vector Spherical Wave (VSW) basis, as an expression for the electrodynamics of a single particle system. This includes the derivation of the T-matrix coefficients for a spherical particle, and an overview of the technique as applied to particles with arbitrarily-shaped boundary surfaces.

Chapter 3 uses considerations based on the radial dependence and angular symmetry of the dipole near fields to develop a set of simple, *qualitative* rules governing the behavior of the wavelength-integrated magnitude of the optical cross sections of nanoparticle aggregates as well as of their spectral properties. These rules provide a guideline both to assist in the interpretation of experimental results and to assist in the design of new experiments.

Chapter 4 discusses the extension of the T-matrix formalism to multiple-particle structures. An overview of the derivation of the *translation* operator for the VSW basis is presented and, in combination with the details of Chapter 2, this in turn leads to the expression of the multiple-particle T-matrix in terms of the T-matrices of the isolated particles.

Chapter 5 discusses the specifics of the numerical code-base developed for programming of the single-particle and multiple-particle T-matrix calculations. This includes discussion of design considerations affecting the structure of the code-

base and includes as well an overview of current capabilities and possible future extensions. This chapter is supported by details presented in Appendix 1.

Chapter 6 applies the unique perspective afforded by the multiple-particle T-matrix technique to the elucidation of deeper considerations affecting the electrodynamic behavior of aggregate structures. Specifically, the interactions between the collective modes and internal per-particle modes are considered, as are details relating to the distinction between non-radiating and radiating components of the inter-particle near-fields. In combination with the simple rules presented in Chapter 3, this analysis allows new insight to be garnered from existing experimental results but is possibly of even greater importance because of its potential contribution to the design of aggregate structures usable in new experimental applications.

Finally, Appendix 1 presents an abridged reference manual for the numerical codes that produced the figures and calculations presented in the dissertation and thus that underlie much of the analysis presented. The inclusion of this reference manual may contribute to the comprehensibility of some of the specifics of that analysis and, in addition, enables the discussion in Chapter 5 to have precise connection to the specific naming conventions used by, and the classes and utility methods provided by, the code-base itself.

Chapter 2

Single-particle electrodynamics

2.1 The material permittivity function

A formal analysis of optical scattering from nanoparticles begins with the electrodynamic constitutive relations:

$$D_i(\omega, \vec{k}) = \epsilon_{ij}(\omega, \vec{k}) E_j(\omega, \vec{k}), \tag{2.1}$$

and

$$B_i(\omega, \vec{k}) = \mu_{ij}(\omega, \vec{k}) H_j(\omega, \vec{k}). \tag{2.2}$$

Most generally, it is asserted that the induced fields $\vec{D}$ and $\vec{B}$ depend on the electric and magnetic fields $\vec{E}$ and $\vec{H}$ at all past times and throughout all of space. Issues of causality must be considered here, but any relativistic considerations are beyond the scope of the discussion. The lack of *temporal* locality is known as *frequency dispersion*, and the lack of *spatial* locality as *spatial dispersion* [43].

Considering Equation 2.1 only, even for isotropic materials the result is a *tensor* permittivity [43]

$$\epsilon_{ij}(\omega, \vec{k}) = \epsilon_T(\omega, k)(\delta_{ij} - k_i k_j / k^2) + \epsilon_L(\omega, k) k_i k_j / k^2, \tag{2.3}$$

where ϵ_T and ϵ_L are the *transverse* and *longitudinal* permittivity respectively.

It can be shown that for particle radii such that $r \gg v_f/\omega$ the effect of the longitudinal permittivity may be neglected. In fact, in this case spatial dispersion may also be neglected (see further discussion in Section 2.2.5). Here v_f is the *Fermi* velocity, and v_f/ω is the characteristic distance traveled by the carriers in one optical cycle. This allows the transverse permittivity to be considered exclusively and the dependence on $\vec{k}$ to be neglected in the Relations 2.1 and 2.2.

For a successful theoretical treatment, some form of dielectric permittivity function must be used in the electrodynamic constitutive relations. For the nonmagnetic materials considered here, only Equation 2.1 will be of concern and 2.2 will be utilized with $\mu = \mu_0$. In this section a dielectric permittivity will be proposed that is sufficient for the present objectives. Key phenomenological features of the permittivity of metals and dielectric materials will be described. For the most part, discussion of the quantum mechanical behavior responsible for these features will be beyond the scope of discussion.

The requirement for causality is enforced by switching to the time domain, and here 2.1 becomes

$$\vec{D}(t) = \epsilon_0 \vec{E}(t) + \epsilon_0 \int_0^\infty \chi(\tau)\vec{E}(t-\tau)d\tau. \tag{2.4}$$

This is the most general causal electrodynamic constitutive relation for the electric fields [43, pg. 266]. It is immediately obvious that, for the time-domain integral to converge, the Fourier transform of the complex susceptibility $\chi(\omega)$ must be analytic in the upper-half ω-plane. The Kramers-Kronig dispersion relations follow directly from this requirement [44, pg. 334]:

$$\Re\epsilon(\omega)/\epsilon_0 = 1 + \frac{2}{\pi}\mathrm{P}\int_0^\infty \frac{\omega'\Im\epsilon(\omega')/\epsilon_0}{\omega'^2 - \omega^2}d\omega', \tag{2.5}$$

$$\Im\epsilon(\omega)/\epsilon_0 = -\frac{2\omega}{\pi}\mathrm{P}\int_0^\infty \frac{\Re\epsilon(\omega')/\epsilon_0 - 1}{\omega'^2 - \omega^2}d\omega'. \tag{2.6}$$

Here P means *Cauchy principle value*. These dispersion relations are important in that they allow the reconstruction of the permittivity function from empirical measurement of the reflectivity alone. To accomplish this objective a closely related form is used[45, pg. 252]:

$$\theta = -\frac{2\omega}{\pi}\mathrm{P}\int_0^\infty \frac{\ln|\mathcal{R}(\omega')|}{\omega'^2-\omega^2}d\omega'. \tag{2.7}$$

Here the angle θ is the phase change in the fields upon reflection at normal incidence. Equation 2.7 relates the complex reflection coefficient $\mathcal{R}$ to the measurement of the reflectivity $|\mathcal{R}|^2$, and therefore can be used through the Fresnel reflectance formulas to reconstruct the permittivity.

2.1.1 A simple model

A simplest model of the susceptibility function $\chi(\omega) = \frac{\epsilon(\omega)}{\epsilon_0} - 1$ for materials treats the behavior of the bound and free electrons separately. The total susceptibility is then the sum of the components.

For bound electrons with a single resonance at $\omega = \omega_i$ of width γ_i and *oscillator strength* f_i the contribution to the susceptibility is [44, pg. 310]

$$\chi_{ib}(\omega) = \frac{f_i}{\omega^2 - \omega_i^2 - i\omega\gamma_i}. \tag{2.8}$$

For free electrons with number density n, Z electrons per atom, and *effective mass* m^* the contribution to the susceptibility is

$$\chi_{fe}(\omega) = -\frac{\omega_p^2}{\omega^2 - i\omega\gamma_e}. \tag{2.9}$$

Here the *plasma frequency* is

$$\omega_p = \frac{nZe^2}{\epsilon_0 m^*}.$$

In Equation 2.9 the effect of the electron mean-free path l_e is implicitly included in the collision frequency term $\gamma_e = v_F/l_e$ [44, pg. 313]. Most generally, an appropriate combination of resonance terms with a free electron term including a suitable collision frequency will accurately model the susceptibility of almost any material.

The effect of the individual terms is as follows [44, pg.310]:

- when ω is sufficiently less than any ω_{ib}, $\chi(\omega)$ is positive;
- as ω passes above a given ω_{ib}, $\chi(\omega)$ decreases in magnitude;
- and finally, when ω is greater than every ω_{ib} and also greater than ω_p, $\chi(\omega)$ is negative.

The possibility of $\chi(\omega) < -1$ will be seen to be a key requirement for resonance scattering (see discussion in Section 2.2.3.1).

Note that for a semiconducting material with an energy gap $E_g = \hbar\omega_g$, for ω sufficiently above ω_g all of the electrons may be treated as free and so, for sufficiently high frequencies, metals and dielectrics behave similarly [45]. Realistically, this does not occur until the x-ray region at $\omega \approx 10^{16} s^{-1}$, or $\lambda \approx 10\text{nm}$.

This simple model of the susceptibility will be used in the discussion of qualitative features of scattering behavior. In addition, the model is used to correct the empirically obtained bulk dielectric permittivity for the effects of particle size comparable to the electron mean-free path l_e. Starting with an empirically obtained dielectric function, the free-electron contribution is extracted using the high-ω asymptotic behavior. This extraction is facilitated by using known values for the plasma frequency ω_p, the electron effective mass m^*, and the Fermi velocity v_F, and its objective is to obtain the collision frequency γ_e. The bulk mean-free path $l_e = v_F/\gamma_e$

is adjusted for the effect of small particle size, and a corrected mean-free path is obtained:

$$\frac{1}{l} = \frac{1}{l_e} + \frac{a}{r}. \tag{2.10}$$

Here a is an empirically determined weighting factor and r is the radius of the particle, or some other appropriate dimension (such as shell thickness). Finally, the dielectric function is re-assembled [29]. This form of dielectric function is utilized in the computer codes discussed in Chapter 5.

2.1.2 Empirical measurement

With respect to empirically measured dielectric functions for the noble metals, the data by Christy et al. [46] are the most widely cited and used. These measurements include data for Ag, Au, and Cu. With the addition of Pt (which has a resonance in the UV [30] and so is not as useful in the current work) these metals are those primarily utilized in biological labeling applications.

The measurement of the dielectric function of a bulk material generally includes the following three steps [45]:

1. A reliable method is found for measuring the reflectivity $\Re$ over the widest possible frequency range.

2. A technique to extend the data at the high and low frequency endpoints is selected.

3. A Kramers-Kronig inversion (Equation 2.7) is used to construct the complex permittivity from the real reflectivity data.

Since essentially all measurements of the dielectric susceptibility include the

Kramers-Kronig inversion step, it is important to consider the mathematical limitations of this step.

In its most general form, a Kramers-Kronig relation specifies a relationship between some desired target function, at a single value of ω, and the integral of another function, over an infinite range of ω. However, any actual experiment necessarily involves measurement over a limited range of ω and can only include a finite number of discrete values.

Analysis by Milton et al. [47] provides a precisely defined set of boundary functions constraining the accuracy of the reconstruction of the target function. For example, the permittivity $\epsilon(\omega) = \epsilon_1(\omega) + i\epsilon_2(\omega)$ may be used, where ϵ_1 and ϵ_2 are real functions with suitable analytic behavior. Measurements of ϵ_2 are taken over a finite range of closely spaced values of ω as well as measurements of ϵ_1 at a few values of ω. Here the appropriate relation is Equation 2.5, written here as

$$\epsilon_1(\omega)/\epsilon_0 = 1 + \frac{2}{\pi}\mathrm{P}\int_0^\infty \frac{\omega'\epsilon_2(\omega')/\epsilon_0}{\omega'^2 - \omega^2}d\omega'.$$

Since the target function ϵ_1 is known at several points, the discrepancy between the target function and its reconstruction must be zero at these points. Using these measured values along with additional requirements on the analytic behavior of the discrepancy function, a hierarchy of rational boundary functions is constructed. At each level of the hierarchy more data points are used, and the boundary constraints for the reconstruction of the target function become more precise.

Provided that the measurements of ϵ_2 are sufficiently close together in ω that no resonances fall between successive ω values, the hierarchy of boundary functions may be used to accomplish several useful objectives. First, error in the reconstruction of ϵ_1 may be assessed. Second, validity of a target ϵ_1 obtained using ϵ_2 data may be tested. (A failure would indicate problems in some aspect of the measurement.)

Third, consistency of ϵ_2 data taken over disjoint ranges of ω, perhaps using different methods, may be evaluated.

It should be noted that so far the permittivity data of Christy et al. [46] have been validated largely by direct comparison with data taken by other groups who have in many cases employed alternative measurement methods. The above analysis should prove useful for processing new measurements of permittivity, for example, when validating measurements as they become available of the longitudinal permittivity over the optical frequency range.

2.2 T-matrix formalism

The most popular approach to the discussion of the single-particle scattering cross section uses the Rayleigh approximation. Although this approximation is certainly appropriate for nanometer-scale particles, it is rather quickly violated during practical application. For example, the requirement that $kr \ll 1$ is clearly violated at $r = 1/k$ which is about $100\mathrm{nm}$ at visible wavelengths. Since particles larger than $r = 80\mathrm{nm}$ are often used in applications of resonance scattering, a completely general theory will be introduced at this point. The small-radius limit of the theory will be shown to be the Rayleigh approximation, and the effects of features of the dielectric permittivity will then be discussed within this limit.

T-matrix formalism, based on the use of the system transfer operator, enables an elegant treatment of electromagnetic scattering. This formalism is completely equivalent to Mie theory when applied to spherical scatterers and is generalizable to allow any incident field with arbitrarily shaped scatterers [26].

The system transfer operator approach is based on the formal specification

$$\vec{E}_s = \mathcal{T}\vec{E}_i.$$

Here the electric field vectors for the incident $\vec{E}_i$ and scattered $\vec{E}_s$ fields are indicated, although the magnetic fields may also be used.

The transfer operator $\mathcal{T}$ is a linear operator, and it is usually expressed in matrix form although care must be used because, in going between $\vec{E}_i$ and $\vec{E}_s$, the fields usually switch between regular and outgoing waves.

2.2.1 Vector spherical wave (VSW) basis

A suitable basis must be chosen for representing the transfer operator. For objects consisting of closed, smooth boundary surfaces, the vector spherical waves (VSW) are often appropriate. Although any type of basis may be used, provided that it is complete, it is important to consider convergence of the numerical representation. For objects with uniform aspect ratio, a spherically symmetric basis is almost always the best choice. With readily available computational hardware and software, using IEEE-754 `double` precision[1], the VSW can be practically implemented up to a multipole order of about $l = 150$, which allows accurate calculation for particle radii r up to about $kr = 100$ [48].

The transverse VSW are the simultaneous eigenfunctions of the transverse vector Helmholtz operator $\{\nabla\times\nabla\times\}$, the total angular momentum operator $\{-\vec{r}\times\nabla\}$ and the azimuthal angular momentum operator $\{-\vec{\rho}\times\nabla\}$. For a given eigenfunction $\vec{M}_{lm}$

$$\begin{aligned}\nabla\times\nabla\times\vec{M}_{lm} &= k^2\vec{M}_{lm},\\ -\vec{r}\times\nabla\vec{M}_{lm} &= l(l+1)\vec{M}_{lm}, \text{ and}\\ -\vec{\rho}\times\nabla\vec{M}_{lm} &= m\vec{M}_{lm}.\end{aligned}$$

[1] IEEE-754 double precision has a 53-bit mantissa, and an 11-bit exponent.

Here $\vec{\rho}$ is the transverse radial vector. With the addition of the analogous eigenfunctions for the longitudinal vector Laplacian operator $\{\nabla\nabla\cdot\}$, the VSW form a complete set of vector functions for radiative fields with a given wave-number. There are many excellent derivations of the explicit structure of these functions [44, 49, 50, 51]. Here the derivation follows [49] and [50].

For the boundary condition requirements of many problems, it is essential to separate the VSW into *transverse* functions $\vec{M}_{lm}$ and $\vec{N}_{lm}$ with $\nabla \cdot \vec{M}_{lm} = 0$, and *longitudinal* functions $\vec{L}_{lm}$ with $\nabla \times \vec{L}_{lm} = 0$. Unfortunately, although *closed* and *complete* under the appropriate Helmholtz operators, this $\{\vec{M}_{lm}, \vec{N}_{lm}, \vec{L}_{lm}\}$ set is not fully orthogonal (see Equation 2.15). For this reason, a two-tiered scheme has been developed for the application of the VSW. This scheme consists of the transverse and longitudinal functions $\{\vec{M}_{lm}, \vec{N}_{lm}, \vec{L}_{lm}\}$ but is expressed in terms of a fully orthonormal set of vector spherical harmonics $\{\vec{P}_{lm}, \vec{B}_{lm}, \vec{C}_{lm}\}$ combined with a suitable radial function. Various normalization conventions are followed in the literature. Here the convention in Tsang [49] is used, but with the spherical harmonics Y_{lm} normalized so that

$$\int_{\Omega} d\Omega Y_{lm}(\hat{n}) Y^*_{l'm'}(\hat{n}) = \delta_{ll'}\delta_{mm'}.$$

In addition, the present scheme normalizes the functions $\{\vec{P}_{lm}, \vec{B}_{lm}, \vec{C}_{lm}\}$. These choices eliminate the usual requirement for the separation of the fields into even and odd components, which separation adds unnecessary complexity to many derivations.

The three *vector spherical harmonic* functions

$$\begin{aligned}
\vec{P}_{lm} &= \hat{r}Y_{lm}(\hat{n}), \\
\vec{B}_{lm} &= \frac{1}{\sqrt{l(l+1)}} r\nabla\left(Y_{lm}(\hat{n})\right) \\
&= \hat{r}\times\vec{C}_{lm}, \text{ and} \\
\vec{C}_{lm} &= -\frac{1}{\sqrt{l(l+1)}}\vec{r}\times\nabla\left(Y_{lm}(\hat{n})\right)
\end{aligned} \tag{2.11}$$

form a complete basis under the angular momentum operators [50, pg. 1900]. Note that only $\vec{P}_{lm}$ has a radial component. In comparison to Jackson [44], the $\vec{C}_{lm} = \vec{X}_{lm}$, where the $\vec{X}_{lm}$ are the vector spherical harmonics as defined by Jackson. These $\vec{X}_{lm}$ are a useful simplification but are problematic in that the longitudinal elements are omitted and therefore the basis is incomplete.

The members of the set $\{\vec{P}_{lm}, \vec{B}_{lm}, \vec{C}_{lm}\}$ satisfy the orthonormality relation

$$\begin{aligned}
\int_\Omega d\Omega \vec{U}_{lm}\cdot\vec{U}^*_{l'm'} &= \delta_{ll'}\delta_{mm'} \\
\int_\Omega d\Omega \vec{U}_{lm}\cdot\vec{V}^*_{l'm'} &= 0,
\end{aligned} \tag{2.12}$$

where $\vec{U}_{lm}$ here represents any member of the set $\{\vec{P}_{lm}, \vec{B}_{lm}, \vec{C}_{lm}\}$, with $\vec{V}_{lm}$ a different member of the same set. The notation $\int_\Omega d\Omega$ designates integration over the solid angle.

The transverse electric (TE) and transverse magnetic (TM) VSW are

$$\begin{aligned}
\vec{M}_{lm} &= z_l(x)\vec{C}_{lm}(\hat{n}) \\
\vec{N}_{lm} &= \sqrt{l(l+1)}\frac{z_l(x)}{x}\vec{P}_{lm}(\hat{n}) + \frac{(xz_l(x))'}{x}\vec{B}_{lm}(\hat{n}),
\end{aligned} \tag{2.13}$$

respectively, and the longitudinal VSW are

$$\vec{L}_{lm} = z_l'(x)\vec{P}_{lm}(\hat{n}) + \sqrt{l(l+1)}\frac{z_l(x)}{x}\vec{B}_{lm}(\hat{n}). \tag{2.14}$$

The radial functions $z_l(x)$ are selected from the set of spherical Bessel and Hankel functions $\{j_l(x), n_l(x), h_l^{(1)}(x), h_l^{(2)}(x)\ldots\}$ as appropriate to the boundary conditions. The normalized radial coordinate is $x = kr$, and the prime symbol indicates differentiation with respect to the argument.

The explicit normalization of the VSW is

$$\begin{aligned}
\int_\Omega d\Omega \vec{M}_{lm}(x,\hat{n}) \cdot \vec{M}^*_{l'm'}(x,\hat{n}) =& \delta_{ll'}\delta_{mm'} \left|z_l(x)\right|^2 \\
\int_\Omega d\Omega \vec{N}_{lm}(x,\hat{n}) \cdot \vec{N}^*_{l'm'}(x,\hat{n}) =& \delta_{ll'}\delta_{mm'} \left\{ \frac{l+1}{2l+1}\left|z_{l-1}(x)\right|^2 + \frac{l}{2l+1}\left|z_{l+1}(x)\right|^2 \right\} \\
\int_\Omega d\Omega \vec{L}_{lm}(x,\hat{n}) \cdot \vec{L}^*_{l'm'}(x,\hat{n}) =& \delta_{ll'}\delta_{mm'} \left\{ \frac{l}{2l+1}\left|z_{l-1}(x)\right|^2 + \frac{l+1}{2l+1}\left|z_{l+1}(x)\right|^2 \right\} \\
\int_\Omega d\Omega \vec{M}_{lm}(x,\hat{n}) \cdot \vec{N}^*_{l'm'}(x,\hat{n}) =& 0 \\
\int_\Omega d\Omega \vec{L}_{lm}(x,\hat{n}) \cdot \vec{M}^*_{l'm'}(x,\hat{n}) =& 0 \\
\int_\Omega d\Omega \vec{L}_{lm}(x,\hat{n}) \cdot \vec{N}^*_{l'm'}(x,\hat{n}) =& \delta_{ll'}\delta_{mm'} \sqrt{l(l+1)} \left\{ 2z_l(x) \frac{(xz_l(x))'^*}{\left|x\right|^2} - \frac{\left|z_l(x)\right|^2}{\left|x\right|^2} \right\}.
\end{aligned} \tag{2.15}$$

In application, the lack of orthogonality between $\vec{L}_{lm}$ and $\vec{N}_{lm}$ is treated either by using the orthonormality of the $\{\vec{P}_{lm}, \vec{B}_{lm}, \vec{C}_{lm}\}$ or through appropriate source current considerations such as localization.

The curl equations for the transverse VSW are:

$$\begin{aligned}
\nabla \times \vec{M}_l &= k\vec{N}_l \\
\nabla \times \vec{N}_l &= k\vec{M}_l.
\end{aligned} \tag{2.16}$$

A necessary addition formula that allows simplification of the scattering cross sec-

tion is [49, pg. 189]

$$\sum_m \left|\hat{\beta}\cdot\vec{C}_{lm}(\hat{n})\right|^2 = \frac{2l+1}{4\pi}P_l'(1)$$
$$= \frac{(2l+1)}{4\pi}\frac{l(l+1)}{2}. \tag{2.17}$$

Here $\hat{\beta}$ is an arbitrary unit vector (e.g. field polarization) and $P_l'(x)$ is the derivative of the Legendre function of the first kind of degree l. This formula can be derived from the addition formula for the Y_{lm} (see Jackson [44, pg. 110]). The relationship between $\vec{B}_{lm}$ and $\vec{C}_{lm}$ in Equations 2.11 implies an analogous formula for the $\vec{B}_{lm}$.

Also required for cross-section calculations is the expansion of an incident plane wave of polarization $\hat{\beta}$ and propagation vector $\vec{k}_i$

$$\vec{E}_i = \hat{\beta}E_0 e^{i\vec{k}_i\cdot\vec{r}}.$$

This expansion can be derived from integral formulas for the vector spherical harmonics of Equations 2.11. The result is [49, pg. 176]:

$$\vec{E}_i(\vec{r}) = a_{lm}^{(i)}\vec{M}_{lm}^{(1)}(x,\hat{r}) + b_{lm}^{(i)}\vec{N}_{lm}^{(1)}(x,\hat{r})$$

where

$$a_{lm}^{(i)} = 4\pi i^l \vec{C}_{lm}^{*}\cdot\hat{\beta}^* E_0, \text{ and}$$
$$b_{lm}^{(i)} = 4\pi i^{l-1}\vec{B}_{lm}^{*}\cdot\hat{\beta}^* E_0. \tag{2.18}$$

Here the superscript (1) indicates use of radial functions that are *regular* at the origin (i.e. $j_l(x)$).

2.2.2 T-matrix for a sphere: Mie scattering theory

This section presents a derivation of the transfer operator for a spherical scatterer. The derivation follows Stratton [52] and Aden [53] but uses the normalization convention discussed above. The derivation is technically equivalent to that of Mie [3].

Note that the Mie solution is often considered exact, but this is not correct for media with a significant free-electron contribution to the dielectric permittivity. This point will be elaborated at the end of the section.

The incident electric field is expanded in vector spherical waves (VSW). Time dependence as $e^{-i\omega t}$ is implicit.

$$\vec{E}_i(\vec{r}) = a_{lm}^{(i)}\vec{M}_{lm}^{(1)} + b_{lm}^{(i)}\vec{N}_{lm}^{(1)}.$$

The magnetic induction is obtained from Faraday's law

$$\begin{aligned}\nabla \times \vec{E}_i &= -\frac{\partial \vec{B}_i}{\partial t} \\ &= i\omega \vec{B}_i\end{aligned}$$

and using Equations 2.16:

$$\begin{aligned}\vec{B}_i(\vec{r}) &= (-i/\omega)\nabla \times \vec{E}_i \\ &= (-ik/\omega)\{b_{lm}^{(i)}\vec{M}_{lm}^{(1)} + a_{lm}^{(i)}\vec{N}_{lm}^{(1)}\}.\end{aligned}$$

The corresponding representations for the internal and scattered fields are

$$\begin{aligned}\vec{E}_t(\vec{r}) &= a_{lm}^{(t)}\vec{M}_{lm}^{(1)} + b_{lm}^{(t)}\vec{N}_{lm}^{(1)}, \\ \vec{B}_t(\vec{r}) &= (-ik_t/\omega)\{b_{lm}^{(t)}\vec{M}_{lm}^{(1)} + a_{lm}^{(t)}\vec{N}_{lm}^{(1)}\}, \\ \vec{E}_s(\vec{r}) &= a_{lm}^{(s)}\vec{M}_{lm}^{(3)} + b_{lm}^{(s)}\vec{N}_{lm}^{(3)}, \text{ and} \\ \vec{B}_s(\vec{r}) &= (-ik/\omega)\{b_{lm}^{(s)}\vec{M}_{lm}^{(3)} + a_{lm}^{(s)}\vec{N}_{lm}^{(3)}\}.\end{aligned}$$

The parameter of the radial functions in the VSW is $x = kr$ where the wave-number k can be k_t or k depending on the region involved. The subscript t indicates the region internal to the sphere and no subscript indicates the surrounding region. The

superscript (1) indicates a radial function regular at the origin ($j_l(x)$). The superscript (3) indicates a radial function asymptotic to e^{ikR}/R as $R \to \infty$ corresponding to an outgoing wave ($h_l^{(1)}(x)$).

For a source free region, the boundary conditions on the tangential components of the fields are

$$\begin{aligned} \hat{r} \times \vec{E}_t &= \hat{r} \times \left(\vec{E}_i + \vec{E}_s \right) \text{ and} \\ (1/\mu_t)\hat{r} \times \vec{B}_t &= (1/\mu)\hat{r} \times \left(\vec{B}_i + \vec{B}_s \right). \end{aligned} \tag{2.19}$$

These tangential components are most easily read from the Equations 2.11 and 2.13, noting that the radial component is contained completely within the $\vec{P}_{lm}$. More formally, this is equivalent to taking the dot product of the fields with $\vec{C}^*_{lm}$ and $\vec{r} \times \vec{C}^*_{lm}$, and performing the integral over the solid angle. Four independent equations are obtained from Equations 2.19, resulting in a solvable system of equations for the coefficients:

$$\begin{aligned} j_l(x_t)a_{lm}^{(t)} - h_l^{(1)}(x)a_{lm}^{(s)} &= j_l(x)a_{lm}^{(i)} \\ (k_t/\mu_t)j_l(x_t)b_{lm}^{(t)} - (k/\mu)h_l^{(1)}(x)b_{lm}^{(s)} &= (k/\mu)j_l(x)b_{lm}^{(i)} \\ \frac{[x_t j_l(x_t)]'}{x_t}b_{lm}^{(t)} - \frac{\left[xh_l^{(1)}(x)\right]'}{x}b_{lm}^{(s)} &= \frac{[xj_l(x)]'}{x}b_{lm}^{(i)} \\ (k_t/\mu_t)\frac{[x_t j_l(x_t)]'}{x_t}a_{lm}^{(t)} - (k/\mu)\frac{\left[xh_l^{(1)}(x)\right]'}{x}a_{lm}^{(s)} &= (k/\mu)\frac{[xj_l(x)]'}{x}a_{lm}^{(i)}. \end{aligned}$$

Note here that, for the derivation of the T-matrix, the incident-field coefficients $a_{lm}^{(i)}$

and $b_{lm}^{(i)}$ are set to unity. The additional simplifications $\mu_t = \mu = \mu_0$ result in

$$\begin{aligned}
j_l(x_t)a_{lm}^{(t)} - h_l^{(1)}(x)a_{lm}^{(s)} &= j_l(x) \\
x_t j_l(x_t)b_{lm}^{(t)} - xh_l^{(1)}(x)b_{lm}^{(s)} &= xj_l(x) \\
x\left[x_t j_l(x_t)\right]' b_{lm}^{(t)} - x_t\left[xh_l^{(1)}(x)\right]' b_{lm}^{(s)} &= x_t\left[xj_l(x)\right]' \\
\left[x_t j_l(x_t)\right]' a_{lm}^{(t)} - \left[xh_l^{(1)}(x)\right]' a_{lm}^{(s)} &= \left[xj_l(x)\right]'.
\end{aligned}$$

These equations are expressed in matrix form as

$$\begin{bmatrix}
-h_l^{(1)}(x) & 0 & j_l(x_t) & 0 \\
0 & -xh_l^{(1)}(x) & 0 & x_t j_l(x_t) \\
0 & -x_t\left[xh_l^{(1)}(x)\right]' & 0 & x\left[x_t j_l(x_t)\right]' \\
-\left[xh_l^{(1)}(x)\right]' & 0 & \left[x_t j_l(x_t)\right]' & 0
\end{bmatrix}
\begin{bmatrix} a_{lm}^{(s)} \\ b_{lm}^{(s)} \\ a_{lm}^{(t)} \\ b_{lm}^{(t)} \end{bmatrix}$$
$$= \begin{bmatrix} j_l(x) \\ xj_l(x) \\ x_t\left[xj_l(x)\right]' \\ \left[xj_l(x)\right]' \end{bmatrix}.$$

Solving for the scattered field:

$$a_{lm}^{(s)} = \frac{\begin{vmatrix}
j_l(x) & 0 & j_l(x_t) & 0 \\
xj_l(x) & -xh_l^{(1)}(x) & 0 & x_t j_l(x_t) \\
x_t\left[xj_l(x)\right]' & -x_t\left[xh_l^{(1)}(x)\right]' & 0 & x\left[x_t j_l(x_t)\right]' \\
\left[xj_l(x)\right]' & 0 & \left[x_t j_l(x_t)\right]' & 0
\end{vmatrix}}{\begin{vmatrix}
-h_l^{(1)}(x) & 0 & j_l(x_t) & 0 \\
0 & -xh_l^{(1)}(x) & 0 & x_t j_l(x_t) \\
0 & -x_t\left[xh_l^{(1)}(x)\right]' & 0 & x\left[x_t j_l(x_t)\right]' \\
-\left[xh_l^{(1)}(x)\right]' & 0 & \left[x_t j_l(x_t)\right]' & 0
\end{vmatrix}}, \text{ and}$$

$$b_{lm}^{(s)} = \frac{\begin{vmatrix} -h_l^{(1)}(x) & j_l(x) & j_l(x_t) & 0 \\ 0 & xj_l(x) & 0 & x_t j_l(x_t) \\ 0 & x_t\left[xj_l(x)\right]' & 0 & x\left[x_t j_l(x_t)\right]' \\ -\left[xh_l^{(1)}(x)\right]' & \left[xj_l(x)\right]' & \left[x_t j_l(x_t)\right]' & 0 \end{vmatrix}}{\begin{vmatrix} -h_l^{(1)}(x) & 0 & j_l(x_t) & 0 \\ 0 & -xh_l^{(1)}(x) & 0 & x_t j_l(x_t) \\ 0 & -x_t\left[xh_l^{(1)}(x)\right]' & 0 & x\left[x_t j_l(x_t)\right]' \\ -\left[xh_l^{(1)}(x)\right]' & 0 & \left[x_t j_l(x_t)\right]' & 0 \end{vmatrix}}.$$

Reverting to T-matrix notation it is noted that:

$$a_{lm}^{(s)} = \mathsf{T}_{lm}^{MM}$$
$$b_{lm}^{(s)} = \mathsf{T}_{lm}^{NN}.$$

Evaluating the determinants results in the T-matrix coefficients:

$$\mathsf{T}_{lm}^{MM} = -\frac{j_l(x_t)\cdot\{xj_l(x)\}' - j_l(x)\cdot\{x_t j_l(x_t)\}'}{j_l(x_t)\cdot\{xh_l^{(1)}(x)\}' - h_l^{(1)}(x)\cdot\{x_t j_l(x_t)\}'}, \text{ and} \tag{2.20}$$

$$\mathsf{T}_{lm}^{NN} = -\frac{x_t^2 j_l(x_t)\cdot\{xj_l(x)\}' - x^2 j_l(x)\cdot\{x_t j_l(x_t)\}'}{x_t^2 j_l(x_t)\cdot\{xh_l^{(1)}(x)\}' - x^2 h_l^{(1)}(x)\cdot\{x_t j_l(x_t)\}'}. \tag{2.21}$$

These coefficients are proportional to the analogous coefficients derived by Mie but are normalized with respect to the multipole content of the incident field. Note that the coefficients are independent of the azimuthal index m, and note also that there is no coupling between the transverse electric $\vec{M}_{lm}$ and transverse magnetic $\vec{N}_{lm}$ fields. This symmetry of the coefficients is to be expected for a spherical scatterer of nonmagnetic material.

2.2.2.1 Truncation criterion

To practically apply the T-matrix for the sphere, a suitable truncation criterion must be specified. The most useful criterion was originally derived by Debye [4] from

the asymptotic behavior of the spherical Hankel functions. In the limit $x \ll l$ these functions behave as [54, pg. 626]

$$j_l(x) \approx \frac{x^l}{(2l+1)!!}, \text{ and} \tag{2.22}$$

$$h_l^{(1)}(x) \approx -i(2l-1)!!x^{-l-1}. \tag{2.23}$$

For $x \gg l$ the functions behave as[ibid.]

$$j_l(x) \approx \frac{1}{x}\sin\left(x - \frac{n\pi}{2}\right), \text{ and} \tag{2.24}$$

$$h_l^{(1)}(x) \approx (-i)^{l+1}\frac{e^{ix}}{x}. \tag{2.25}$$

It follows immediately that for l much larger than the radius of the sphere in optical units, the T_{lm} become exponentially small. Given a numerical calculation of the coefficients for a representative sphere, this behavior is illustrated in Figure 2.1.

Two aspects of this behavior are particularly useful for an intuitive understanding of it. The first is that the radius of the sphere has direct analogy to the *impact parameter* for scattering collisions in classical mechanics. For a collision, the amount of angular momentum that can be transferred scales directly with the impact parameter. The second aspect is that *radiating* fields in a homogeneous medium cannot support detail with dimension significantly less than the wavelength, over optical length-scales greater than the highest multipole order contributing to the fields. When referred to the fields at the surface of the sphere, this corresponds to the mathematical requirement that the coefficients for high l-order VSW must be negligible.

A modern computational evaluation due to Wiscombe[55] provides precise formulas specifying the truncation multipole order that should be used in numerical work:

$$l_{\max} = \begin{cases} x + 4x^{1/3} + 1 & 0.02 \le x \le 8 \\ x + 4.05x^{1/3} + 2 & 8 < x < 4200 \\ x + 4x^{1/3} + 2 & 4200 \le x \le 20000 \end{cases}, \tag{2.26}$$

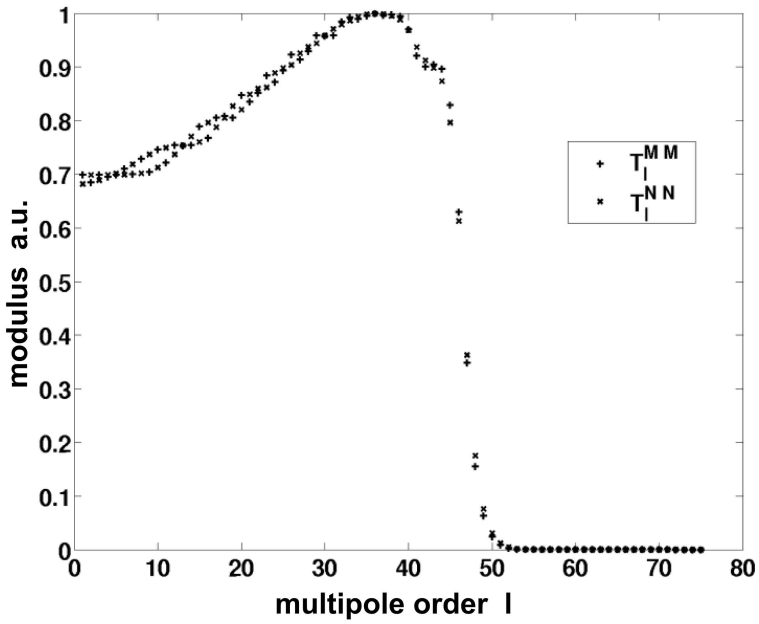


Figure 2.1: Modulus of T-matrix coefficients for a dielectric sphere of radius $x_t = 50$.

where $x = kr$ is the particle radius in optical units and $l_{\max}$ is the maximum multipole order required to achieve IEEE-754 `double` precision accuracy. The specific truncation order specified by these formulas is only slightly influenced by specifics of the dielectric permittivity of the particle.

2.2.3 Extinction and scattering cross sections

For many applications, the optical intensity is the known parameter and the fields are measured only indirectly. Calculation of the intensity requires a relationship between the T-matrix coefficients and the extinction and scattering cross sections.

Given an incident plane wave of unit intensity and polarization $\hat{\beta}$, the absorption cross section $\sigma_{a\beta}$ is the rate of energy loss within a surface $\mathcal{S}$ external to but com-

pletely enclosing the scattering object. It is convenient to define the surface $\mathcal{S}$ as a sphere centered at the origin of the scatterer and with an arbitrarily large radius:

$$\sigma_{a\beta} = -\frac{1}{I_0}\int_{\mathcal{S}} \langle \vec{S} \rangle \cdot \hat{r} ds. \tag{2.27}$$

Here the time-averaged Poynting vector is

$$\langle \vec{S} \rangle = \frac{1}{2}\Re\left(\vec{E} \times \vec{H}^*\right).$$

As $\langle \vec{S} \rangle$ involves the total fields, the scattering and extinction cross sections are implicit in Equation 2.27.

$$-\int_{\mathcal{S}} \langle \vec{S} \rangle \cdot \hat{r} ds = \int_{\mathcal{S}} \langle \vec{S}_i \rangle \cdot \hat{r} ds + \int_{\mathcal{S}} \langle \vec{S}_s \rangle \cdot \hat{r} ds - \int_{\mathcal{S}} \langle \vec{S}_e \rangle \cdot \hat{r} ds.$$

Here

$$\langle \vec{S}_i \rangle = \frac{1}{2}\Re\left(\vec{E}_i \times \vec{H}_i^*\right), \tag{2.28}$$

$$\langle \vec{S}_s \rangle = \frac{1}{2}\Re\left(\vec{E}_s \times \vec{H}_s^*\right), \text{ and} \tag{2.29}$$

$$\langle \vec{S}_e \rangle = \frac{1}{2}\Re\left(\vec{E}_i \times \vec{H}_s^* + \vec{E}_s \times \vec{H}_i^*\right). \tag{2.30}$$

Using the divergence theorem

$$\int_{\mathcal{S}} \langle \vec{S}_i \rangle \cdot \hat{r} ds = \int_{\mathcal{V}} \nabla \cdot \langle \vec{S}_i \rangle d^3x = 0, \tag{2.31}$$

given that a transverse plane wave field has constant energy density.

The scattering and extinction cross sections $\sigma_{s\beta}$ and $\sigma_{e\beta}$ follow the form of Equation 2.27 with the substitution of $\vec{S}_s$ or $\vec{S}_e$ as appropriate.

In order to illustrate the application of the VSW basis, $\sigma_{e\beta}$ will be calculated explicitly. The incident and scattered fields are:

$$\begin{aligned}
\vec{E}_i(\vec{r}) &= a_{lm}^{(i)}\vec{M}_{lm}^{(1)}(k_i r,\hat{r}) + b_{lm}^{(i)}\vec{N}_{lm}^{(1)}(k_i r,\hat{r}) \\
\vec{H}_i(\vec{r}) &= (-ik/\mu_0\omega)\{b_{lm}^{(i)}\vec{M}_{lm}^{(1)} + a_{lm}^{(i)}\vec{N}_{lm}^{(1)}\}, \text{ and} \\
\vec{E}_s(\vec{r}) &= a_{lm}^{(s)}\vec{M}_{lm}^{(3)}(k_i r,\hat{r}) + b_{lm}^{(s)}\vec{N}_{lm}^{(3)}(k_i r,\hat{r}) \\
\vec{H}_s(\vec{r}) &= (-ik/\mu_0\omega)\{b_{lm}^{(s)}\vec{M}_{lm}^{(3)} + a_{lm}^{(s)}\vec{N}_{lm}^{(3)}\}.
\end{aligned}$$

Here the (1) superscript indicates regular fields proportional to $j_l(x)$ and the (3) superscript indicates outgoing fields $h_l^{(1)}(x)$[2].

The integrand for the extinction cross section is

$$\begin{aligned}
\langle\vec{S}_e\rangle\cdot\hat{r} =& \frac{1}{2}\Re\left(\left[\vec{E}_i\times\vec{H}_s^*\cdot\hat{r}\right]_{\mathrm{I}} + \left[\vec{E}_s\times\vec{H}_i^*\cdot\hat{r}\right]_{\mathrm{II}}\right) \qquad (2.32)\\
=& \frac{1}{2}\Re\left(\left[\left(\hat{r}\times\vec{E}_i\right)\cdot\vec{H}_s^*\right]_{\mathrm{I}} + \left[\left(\hat{r}\times\vec{E}_s\right)\cdot\vec{H}_i^*\right]_{\mathrm{II}}\right)\\
[\ldots]_{\mathrm{I}} =& \sum_{lm}\sum_{l'm'}\left((-ik/\mu_0\omega)a_{lm}^{(i)}a_{l'm'}^{(s)*}\left(\hat{r}\times\vec{M}_{lm}\right)\cdot\vec{N}_{l'm'}^*\right.\\
&\left.+(-ik/\mu_0\omega)b_{lm}^{(i)}b_{l'm'}^{(s)*}\left(\hat{r}\times\vec{N}_{lm}\right)\cdot\vec{M}_{l'm'}^*\right) \qquad (2.33)\\
=& \sum_{lm}\sum_{l'm'}\left((-ik/\mu_0\omega)a_{lm}^{(i)}\mathsf{T}_{l'm'}^{MM*}a_{l'm'}^{(i)*}\left(\hat{r}\times\vec{M}_{lm}\right)\cdot\vec{N}_{l'm'}^*\right.\\
&\left.+(-ik/\mu_0\omega)b_{lm}^{(i)}\mathsf{T}_{l'm'}^{NN*}b_{l'm'}^{(s)*}\left(\hat{r}\times\vec{N}_{lm}\right)\cdot\vec{M}_{l'm'}^*\right).
\end{aligned}$$

In Equation 2.33 only the non-zero terms have been retained, and only the derivation for term (I) is shown explicitly. This step ends the first tier of the normalization scheme: the $\{\vec{M}_{lm},\vec{N}_{lm},\vec{L}_{lm}\}$ basis has been used to apply the boundary conditions on the transverse components of the fields. Note that a large part of this step includes the calculation of the T-matrix for the sphere in Section 2.2.2.

[2]To avoid confusion, it is additionally noted that the spherical Hankel functions $h_l^{(1)}(x)$ and $h_l^{(2)}(x)$ also carry superscripts, denoting the *kind* of Hankel function involved, corresponding to *outgoing wave* and *incoming wave* boundary conditions respectively.

The T-matrix for the sphere has the symmetry:

$$\begin{aligned}
\mathsf{T}^{MN} =& 0 \\
\mathsf{T}^{NM} =& 0 \\
\mathsf{T}^{MM}_{lml'm'} =& \mathsf{T}^{MM}_{l} \delta_{ll'} \delta_{mm'} \\
\mathsf{T}^{NN}_{lml'm'} =& \mathsf{T}^{NN}_{l} \delta_{ll'} \delta_{mm'}
\end{aligned}$$

and the coefficients for the scattered field become:

$$\begin{aligned}
a^{(s)}_{lm} &= \sum_{lm} \sum_{l'm'} \mathsf{T}^{MM}_{lml'm'} a^{(i)}_{l'm'} \\
&= a^{(i)}_{lm} \mathsf{T}^{MM}_{l}, \text{ and} \\
b^{(s)}_{lm} &= b^{(i)}_{lm} \mathsf{T}^{NN}_{l}.
\end{aligned}$$

For the sake of brevity, additional sums for coupling between $\vec{M}_{lm}$ and $\vec{N}_{lm}$ and for the transition between *regular* and *outgoing* fields are omitted. In general, such sums would be required for a numerical implementation.

In order to apply orthonormality at this point, the $\{\vec{M}_{lm}, \vec{N}_{lm}\}$ are re-expressed in terms of the $\{\vec{P}_{lm}, \vec{B}_{lm}, \vec{C}_{lm}\}$. In addition, since $x \to \infty$ on the spherical surface $\mathcal{S}$, large-parameter asymptotes for the spherical Bessel functions are used:

$$h_l(x) \to i^{-l-1} \frac{1}{x} e^{ix} \tag{2.34}$$

$$\frac{(x h_l(x))'}{x} \to i^{-l} \frac{1}{x} e^{ix}. \tag{2.35}$$

This means that the radial terms combine to unity for $x \in \mathcal{S}$.

Equation 2.33 becomes

$$\begin{aligned}
[\ldots]_{\mathrm{I}} = \sum_{lm} \Bigg(& (-ik/\mu_0 \omega x^2) a^{(i)}_{lm} \mathsf{T}^{MM*}_{l} a^{(i)*}_{lm} \frac{1}{l(l+1)} \vec{B}_{lm} \cdot B^{*}_{lm} \\
& -(-ik/\mu_0 \omega x^2) b^{(i)}_{lm} \mathsf{T}^{NN*}_{l} b^{(i)*}_{lm} \frac{1}{l(l+1)} \vec{C}_{lm} \cdot C^{*}_{lm} \Bigg).
\end{aligned}$$

The explicit expressions for the incident plane wave are substituted from Equation 2.18:

$$[\ldots]_{\mathrm{I}} = \sum_{lm} \Bigg((-ik/\mu_0\omega x^2) 4\pi^2 \left| \hat{\beta} \cdot \vec{B}_{lm}(\hat{k}_i) \right|^2 \mathsf{T}_l^{MM*} \frac{1}{l(l+1)} \vec{B}_{lm} \cdot B_{lm}^* \\ -(-ik/\mu_0\omega x^2) 4\pi^2 \left| \hat{\beta} \cdot \vec{C}_{lm}(\hat{k}_i) \right|^2 \mathsf{T}_l^{NN*} \frac{1}{l(l+1)} \vec{C}_{lm} \cdot C_{lm}^* \Bigg) .$$

Using the sum rule from Equation 2.17 this becomes

$$[\ldots]_I = \sum_{\mathrm{I}} \left((-2\pi ik/\mu_0\omega x^2)(2l+1)\mathsf{T}_l^{MM*} + (2\pi ik/\mu_0\omega x^2)(2l+1)\mathsf{T}_l^{NN*} \right) .$$

The second term from Equation 2.32 is added and the result is integrated over the solid angle. After normalizing by I_0, the final expression for the cross section is

$$\sigma_{e\beta} = -\frac{2\pi}{k^2} \sum_l (2l+1) \Re \left\{ \mathsf{T}_l^{MM} + \mathsf{T}_l^{NN} \right\} . \tag{2.36}$$

The analogous expression for the scattering cross section is

$$\sigma_{s\beta} = \frac{2\pi}{k^2} \sum_l (2l+1) \left\{ \left| \mathsf{T}_l^{MM} \right|^2 + \left| \mathsf{T}_l^{NN} \right|^2 \right\} , \text{ with} \tag{2.37}$$

$$\sigma_{a\beta} = \sigma_{e\beta} - \sigma_{s\beta}. \tag{2.38}$$

2.2.3.1 Small radius limit

The asymptotic limits of Equations 2.22 and 2.23 also constrain the behavior of the T-matrix for a very small sphere. Applying these expressions in Equations 2.20 and 2.21 gives

$$\mathsf{T}_{lm}^{MM} \to 0$$

$$\mathsf{T}_{lm}^{NN} \to i \frac{(l+1)(2l+1)}{[(2l+1)!!]^2} x^{2l+1} \frac{\epsilon_s - \epsilon}{l\epsilon_s + (l+1)\epsilon}.$$

The T_{lm}^{NN} for $l = 1$ correspond to dipole scattering, and the substitution of this expression in Equations 2.37 and 2.38 results in the Rayleigh scattering approximation:

$$\sigma_{sR} = \frac{8\pi}{3} \frac{x^6}{k^2} \left| \frac{\epsilon_t - \epsilon}{\epsilon_t + 2\epsilon} \right|^2 \tag{2.39}$$

$$\sigma_{aR} = \frac{4\pi}{k^2} x^3 \Im \left\{ \frac{\epsilon_t - \epsilon}{\epsilon_t + 2\epsilon} \right\}. \tag{2.40}$$

An important distinction applies to this approximation. The Rayleigh approximation is always applicable for nonmagnetic particles provided the radius is sufficiently small [2]. There may be a range of particle radii where the quadrupole term is still negligible in comparison to the dipole term but the radial correction to the dipole term accounts for a significant difference from the Rayleigh limit. For this range of radii, although the Rayleigh approximation produces inaccurate results, the *dipole* approximation is still applicable. Either the expansion of the electric dipole term to the next order in r, or the evaluation of the complete term using a numerical implementation of the spherical Bessel functions, may be used to obtain accurate results. This transition away from the Rayleigh approximation is illustrated in Figure 2.2.

With respect to small particles, it is now possible to summarize the effect of the form of the dielectric permittivity as discussed in Section 2.1.1. It is immediately seen that under no circumstances are there resonances present for the magnetic T_{lm}^{MM} terms. In order for there to be resonances in the electric T_{lm}^{NN} terms, the permittivity must be negative, and this corresponds to the free-electron portion of the permittivity. In other words, for small particles either high-ω or a metallic material is required for optical resonances to exist. Finally, it can be seen that, in addition to the dipole resonances, higher-order resonances quickly become important when $x > 1$ with details depending on the specific material. Silver is a material where

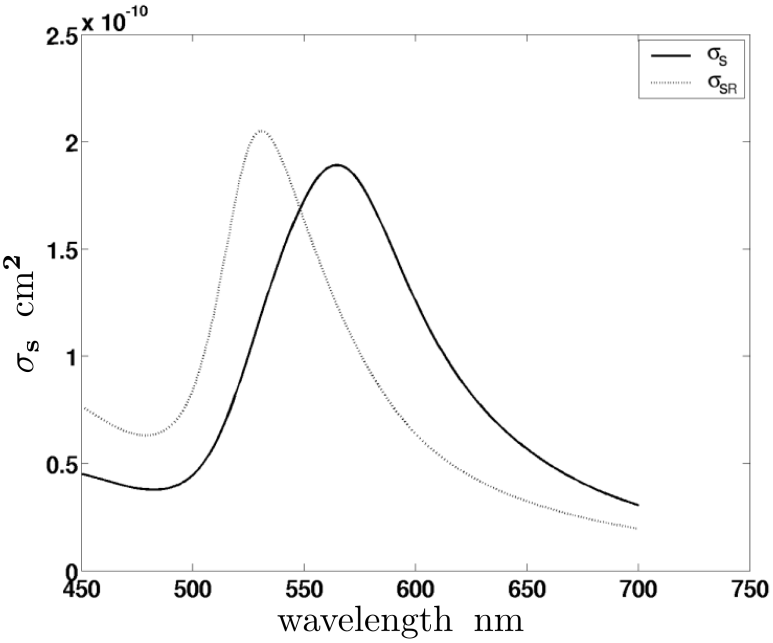


Figure 2.2: Comparison between calculations of the Rayleigh scattering cross section and the Mie cross section for Au ($x = 0.5$). For particles of this size, the difference between the two cross sections is entirely due to the radial dependence of the dipole term.

higher order resonances are readily seen. The onset of the quadrupole resonance for Ag is shown in Figure 2.3.

2.2.4 Arbitrary boundary surfaces

For objects described by *arbitrary* albeit *closed* boundary surfaces a further generalization of the T-matrix calculation is possible. This generalization was initially proposed by Waterman [23] (that is, in the same publication largely responsible for introduction of the T-matrix formalism itself). The approach involves the expression, of the fields internal to the object and of the fields scattered by the object, within two regions defined by spherical boundary surfaces. The internal fields

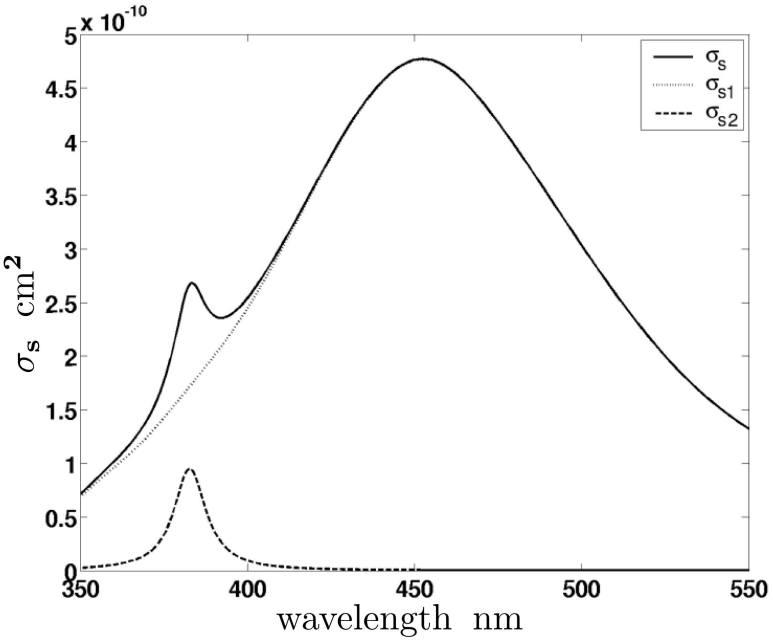


Figure 2.3: Calculation of the Mie scattering cross section for Ag ($x = 0.5$) showing sum of dipole and quadrupole contributions.

are expressed within a spherical boundary completely contained within the actual boundary surface of the object, and the scattered fields are expressed outside of a co-centered spherical boundary circumscribing the object's boundary. Green's function methods are then used to match the boundary conditions on the continuity of the in-surface components of the fields at the actual surface, using hypothetical source-current distributions on the spherical surfaces. This technique is known as the *extended boundary condition* (EBC) method.

Debye's analysis of the dependence of the Mie coefficients on particle radius[4] also applies directly to the T-matrix coefficients for particles defined by arbitrarily-shaped boundary surfaces[24]. Analogously to what is done for the Mie coefficients, this T-matrix-coefficient dependence is demonstrated through consideration

of the magnitude of the *irregular* portion of the VSW basis functions at the object's boundary surface but, in the present case, this constrains the magnitude of the integrands used during the integration steps of the EBC method (that is, steps required for application of the Green's function technique).

A fact worth special notice is that, using this more general T-matrix technique, analytic cross-section formulas are available for the orientationally-averaged cross sections [49, 56]

$$\langle C_s \rangle = \frac{2\pi}{k^2} \|T\|_2^2 \tag{2.41}$$

$$\langle C_e \rangle = -\frac{2\pi}{k^2} \Re \mathrm{Tr}(T) \tag{2.42}$$

where the expression $\|T\|_2^2$ indicates the sum of the squared modulus of all of the elements of the T-matrix (i.e., the square of the *Frobenius* norm) and the expression $\mathrm{Tr}(T)$ indicates the *trace* of the T-matrix. Involving as it does the *trace* of the T-matrix, Formula 2.42 is particularly useful because it enables incremental monitoring of the degree of convergence achieved during the cluster-centered T-matrix calculation (see Section 4.4.2). Formula 2.41 as initially proposed by Mishchenko[56] for the orientationally-averaged scattering cross section was limited to objects with *azimuthal* symmetry, but it was later shown by Khlebtsov[57] to apply without this restriction. These *analytic* expressions for orientationally-averaged cross sections represent a unique advantage of the T-matrix technique, one that facilitates high-speed computational evaluation of such cross sections.

Further discussion of this more general T-matrix technique is beyond the scope of this dissertation except to note that an excellent review of it is presented in Mischenko [26].

2.2.5 Limitations of the boundary conditions

Derivation of the T-matrix coefficients in preceding sections assumed boundary conditions such that the in-surface components of the fields were continuous across boundary surfaces. In this section an alternative to this assumption will be discussed and constraints governing this assumption will be briefly explored.

For media with a significant free-electron contribution to the permittivity, in addition to the *transverse* VSW *longitudinal* VSW must also be included within the sphere [58]. When the internal fields are expanded, an additional boundary condition requires the continuity of the radial component of the displacement current [59]. This adds a slight additional complexity to the derivation of the T-matrix coefficients for a sphere, which is otherwise analogous to the derivation just presented. The results (see Ruppin [58]) are

$$\mathsf{T}_{lm}^{NN} = -\frac{c_l j_l(x) + j_l'(x_L)\left\{x_t^2 j_l(x_t)\cdot\{x j_l(x)\}' - x^2 j_l(x)\cdot\{x_t j_l(x_t)\}'\right\}}{c_l h_l^{(1)}(x) + j_l'(x_L)\left\{x_t^2 j_l(x_t)\cdot\{x h_l^{(1)}(x)\}' - x^2 h_l^{(1)}(x)\cdot\{x_t j_l(x_t)\}'\right\}}, \tag{2.43}$$

where

$$c_l = l(l+1)\left[j_l(x_L)/x_L\right] j_l(x_t)\left(x_t^2 - x^2\right), \tag{2.44}$$

with the T_{lm}^{MM} unmodified.

The longitudinal wave-number k_L solves the dispersion relation [58]

$$1 - \frac{\omega_p^2}{\omega(\omega + i/\tau)}\frac{3}{a^2}\left(1 - \frac{\tan^{-1}(a)}{a}\right) \Big/ \left(1 + \frac{i}{\omega\tau}\left(1 - \frac{\tan^{-1}(a)}{a}\right)\right) = 0, \tag{2.45}$$

where

$$a^2 \stackrel{\text{def}}{=} \frac{-k_L^2 v_F^2}{(\omega + i/\tau)^2},$$

ω_p and v_F are the *plasma frequency* and *Fermi velocity*, and τ is a phenomenological relaxation time.

To evaluate the effect of the inclusion of longitudinal VSW, Relation 2.45 is solved numerically over the optical frequency range. Characteristic results from this calculation are shown in Table 2.1 for a frequency ω_0 corresponding to a free-space wavelength of $\lambda_0 = 532$nm . Values of the plasma frequency ω_p and the Fermi velocity v_F are taken from Aschroft et al. [60]. A conservative relaxation time $\tau = 100/\omega_p$ is used (see discussion in [58]). The concept of *skin depth* is applicable here and refers to the mean penetration depth for modes of a given type. In the present situation, it is seen that the longitudinal skin depth is significantly less than the wavelength, whereas the transverse skin depth is on the order of the wavelength and typically much larger than the particle radius.

Table 2.1: Calculated values for longitudinal wave-number.

material	ω_p/ω_0	v_F/ω_0(Å)	$k_L(m^{-1})$	k_L/k_0
Cu	3.80	4.43	$(1.14 \times 10^7, 1.29 \times 10^{10}i)$	$(0.96, 1.09 \times 10^3 i)$
Ag	3.94	3.93	$(1.27 \times 10^7, 1.52 \times 10^{10}i)$	$(1.08, 1.29 \times 10^3 i)$
Au	3.89	3.95	$(1.27 \times 10^7, 1.49 \times 10^{10}i)$	$(1.07, 1.26 \times 10^3 i)$

At optical frequencies the longitudinal wave-number k_L is almost purely imaginary and is many orders of magnitude greater than the transverse wave-number k_T. This allows the use of the large-parameter asymptotic expression for $j_l(x_L)/x_L$. Thus $c_l \propto (k_0/k_L)^2$ and the effect of the term in Equation 2.44 can be neglected. For particle radii greater than about 1 nm, the T_{lm}^{NN} reduces to the traditional Mie coefficient. Physically this corresponds to the fact that for $\omega < \omega_p$ the longitudinal plasma modes are evanescent and do not penetrate into the sphere sufficiently to cause significant modification of the internal fields. Under the given assumptions the Mie approximation is acceptable, but for very small radius particles or for very short wavelengths it is not.

2.2.6 Core-shell composite particles

For biomedical engineering applications of resonant scattering, it is often useful to be able to adjust the wavelength position of the resonance. To this end a useful extension of the Mie theory involves the analysis of scattering from a core-shell system. The derivation involves the additional expansion of the fields in the shell and consequently more equations are involved, but otherwise the boundary conditions and the solution technique are identical to those presented above. For this system, the results (from Aden [53]) are

$$\mathsf{T}_{lm}^{MM} = -\frac{j_l(x)A_3 + \{xj_l(x)\}'A_4}{h_l^{(1)}(x)A_3 + \{xh_l^{(1)}\}'A_4}, \text{ and} \tag{2.46}$$

$$\mathsf{T}_{lm}^{NN} = -\frac{\{xj_l(x)\}'A_1 + j_l(x)A_2}{\{xh_l^{(1)}(x)\}'A_1 + h_l^{(1)}(x)A_2}, \tag{2.47}$$

where

$$\begin{aligned} A_1 = {} & n_2^2\{x_{11}j_l(x_{11})\}'\left[j_l(x_{22})h_l^{(1)}(x_{21}) - j_l(x_{21})h_l^{(1)}(x_{22})\right] \\ & + n_1^2 j_l(x_{11})\left[\{x_{21}j_l(x_{21})\}'h_l^{(1)}(x_{22}) - \{x_{21}h_l^{(1)}(x_{21})\}'j_l(x_{22})\right] \end{aligned} \tag{2.48}$$

$$\begin{aligned} A_2 = {} & \{x_{11}j_l(x_{11})\}'\left[j_l(x_{21})\{x_{22}h_l^{(1)}(x_{22})\}' - h_l^{(1)}(x_{21})\{x_{22}j_l(x_{22})\}'\right] \\ & + \left(n_1^2/n_2^2\right) j_l(x_{11}) \\ & \times \left[\{x_{22}j_l(x_{22})\}'\{x_{21}h_l^{(1)}(x_{21})\}' - \{x_{21}j_l(x_{21})\}'\{x_{22}h_l^{(1)}(x_{22})\}'\right] \end{aligned} \tag{2.49}$$

$$\begin{aligned} A_3 = {} & j_l(11)\left[\{x_{22}j_l(x_{22})\}'h_l^{(1)}(x_{21}) - \{x_{21}h_l^{(1)}(x_{21})\}'j_l(x_{22})\right] \\ & + \{x_{11}j_l(11)\}'\left[\{x_{21}j_l(x_{21})\}'h_l^{(1)}(x_{22}) - \{x_{21}h_l^{(1)}(x_{21})\}'j_l(x_{22})\right] \end{aligned} \tag{2.50}$$

$$\begin{aligned} A_4 = {} & j_l(11)\left[\{x_{21}j_l(x_{21})\}'h_l^{(1)}(x_{22}) - \{x_{21}h_l^{(1)}(x_{21})\}'j_l(x_{22})\right] \\ & + \{x_{11}j_l(11)\}'\left[j_l(x_{22})h_l^{(1)}(x_{21}) - j_l(x_{21})h_l^{(1)}(x_{22})\right] \end{aligned} \tag{2.51}$$

and $x_{ij} = n_i k r_j$. Numbered subscripts correspond to Regions 1 through 3 as shown in Figure 2.4. This analysis has been extended to allow an arbitrary number of spherical shells and is presented in a recursive form in Sinzig [61].

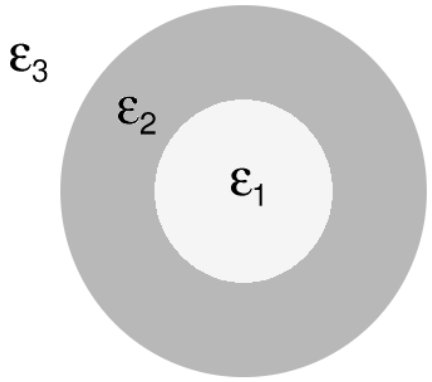


Figure 2.4: Region enumeration for core-shell particle.

When applied to a shell of noble metal over a dielectric core, the core-shell system allows the adjustment of the wavelength of the maximum scattering cross section throughout the visible range. An example of such an adjustment is shown in Figure 2.5 for the $\mathrm{Ag} : \mathrm{SiO_2}$ system.

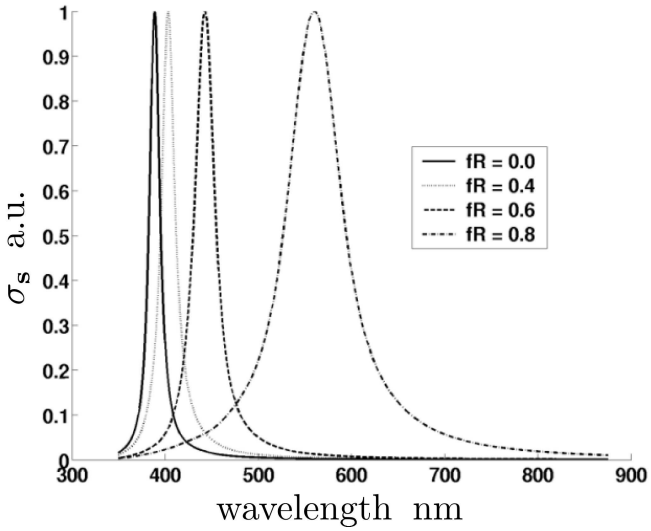


Figure 2.5: Scattering cross section for $\mathrm{Ag} : \mathrm{SiO_2}$ core-shell particle ($r = 20\mathrm{nm}$ $f_R = r_\mathrm{c}/r_\mathrm{s}$). Cross sections are rescaled for comparison.

Chapter 3

Phenomenology

3.1 Introduction

The analysis of this dissertation is motivated by a particular real-world problem. When molecular targets of biological interest are labeled with plasmonic resonant nanoparticles, target co-localization on nanometer-length scales induces observable changes in the scattering spectra. More specifically, the plasma membrane of biological cells includes sub-domains with size on the order of 100nm and cumulative area of about 10% of the total membrane area [62]. Targets of interest for nanoparticle labeling such as the epithelial growth-factor receptor (EGFR) are associated with these sub-domains [63, 64]. For such labeling applications, a typical nanoparticle aggregate size would be 100nm with about 150nm distance between aggregate centers. Nanoparticles used for labeling are essentially spherical and have a diameter of about 20nm .

These factors motivate the analysis of a planar system with multiple length-scales, a large length-scale on the order of $50/k$ (corresponding to the size of the entire cell), and a small length-scale of about $1/k$ (corresponding to the size of the membrane sub-domains). Here k is the wave-number $2n_m\pi/\lambda$ corresponding to the visible or near-infrared (NIR) optical wavelength λ used for observation, with n_m the refractive index of the medium surrounding the nanoparticles. This application also motivates a working definition of *aggregate* as noted below. In general, an experimental observation of an optical cross section from a nanoparticle aggregate implic-

itly includes some form of integration over angular and polarization components. For many biomedical applications, apart from the existence of depolarization, the details of angular symmetry effects in the scattering cross section are not particularly important and will therefore not be discussed in detail here. It will be noted, however, that significant additional information would be available to any imaging or non-imaging modalities able to make use of this angular and polarization information.

A typical experiment is illustrated schematically in Figure 3.1. Gold nanoparticles are conjugated to anti-EGFR antibody and used to label live A431 cells. In response to epithelial growth factor (EGF) added to the growth medium, and to a certain degree to antibody binding itself, these cell-surface receptors are endocytosed by the cells. Several distinct points in the endocytosis sequence are isolated either through the use of time-sequencing or through temperature control of the cell growth medium. Optical spectra are resolved in real-time using a dark-field microscope equipped with a hyper-spectral imaging system (a Leica DM6000 microscope with PARISS® hyper-spectral imaging device). See Figure 3.2 for examples of such spectra[1]. Specifics of nanoparticle aggregate morphology at these time points are established via formalin fixation followed by epoxy embedding, micro-tome slicing, and examination under a transmission electron microscope (a Philips EM208 TEM). See Figure 3.3 for image examples. For additional verification, direct correspondence between images from TEM and images from optical hyper-spectral microscopy has also been successfully demonstrated with epoxy slices. Details of this and of similar results are published elsewhere [22, 65, 66].

Although it is relatively straightforward to calculate the optical cross section of any

[1]The experimental optical spectra and TEM images used in this section are courtesy of Jesse Aaron and Nathan Harrison, respectively.

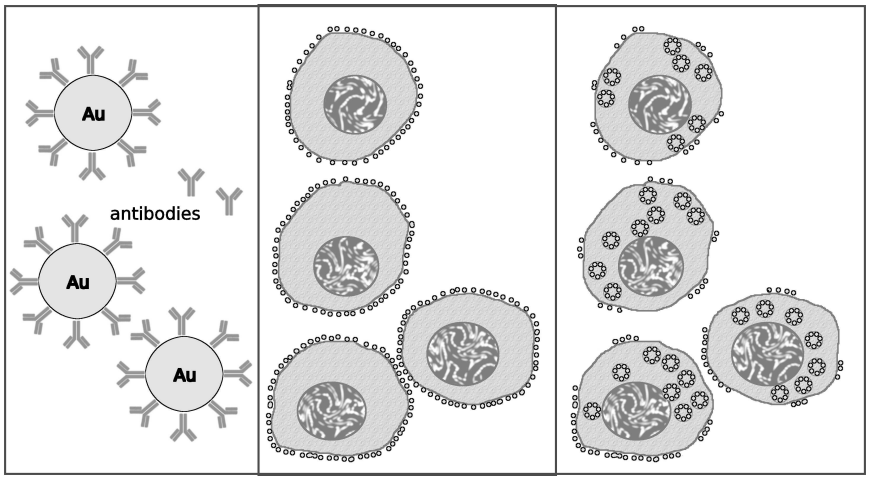


Figure 3.1: Schematic of live-cell labeling experiment showing gold nanoparticle conjugation to anti-EGFR, labeling of cell-surface receptors, and endocytosis of labeled receptors.

nanoparticle aggregate with well-defined particle positions, this is not the case for reconstructing an aggregate's spatial distribution from its integrated cross section. Most often, there is insufficient information remaining for this inverse problem to be well-posed. That is, it is not possible to calculate which specific aggregate, with which specific inter-particle positions, produced a measured cross section. On the other hand, it is possible to quantify statistical features of cross sections produced by aggregates falling into suitably limited classes, and the limitations of these statistical features can be used to extract useful experimental information from cross-section measurements. Given as specific examples in the discussion to follow, these classes of aggregates are limited by restricting inter-particle spacing, total particle number, aspect ratio, overall dimensionality, and details of the pair-correlation function affecting particle packing within the aggregate. Additional or alternative restrictions on aggregate morphology may be possible depending on experiment specifics.

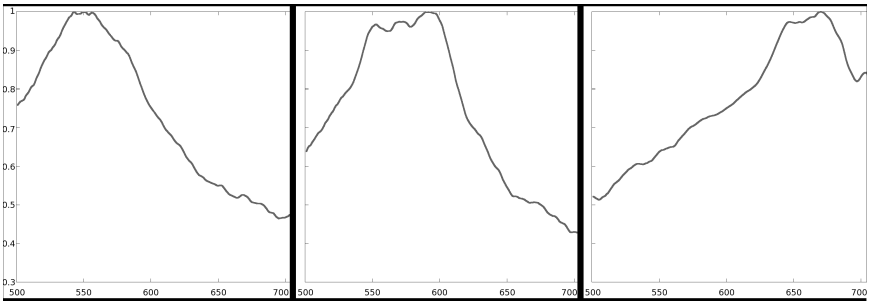


Figure 3.2: Scattering spectra of nanoparticle aggregates from cells in $5°$C, $25°$C, and $37°$C media-temperature-control experiments (ordinate: normalized scattering cross section, abscissa: wavelength in nanometers). Temperature points correspond to early, intermediate, and late endosome stages.

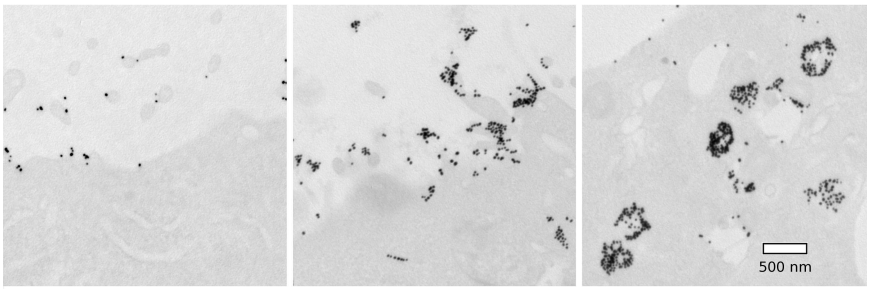


Figure 3.3: TEM images from fixed, epoxy-embedded cell slices at experimental conditions corresponding to those of Figure 3.2.

In the evaluation of scattering from nanoparticle aggregates comprised of a large number of particles, it is desirable to use techniques allowing elucidation of qualitative trends in the resonance spectra. In particular, the use of the calculation for an infinite lattice enables understanding the possible extremes of spectral modification. Then, smaller lattices can be evaluated to illustrate behavior of the transition from a few particles to the largest aggregates. As the last step in the analysis, effects of random inter-particle distance and angle will be briefly discussed.

The topic of scattering from nanoparticle aggregates is complex. For example, sophisticated diagrammatic formalism, including Feynman diagrams analogous to

those used for quantum field theory, has been usefully applied to it. The requirement for something like this formalism stems from the fact that the number of possible intra-aggregate scattering event types scales with the factorial of the number of particles in the aggregate. In its general scope this problem becomes tractable only through careful classification of intra-aggregate scattering events and then, for each resulting class, an estimation of the importance of its contribution to the cross section [32, 33]. By considering a limited version of the problem, the difficulties of this approach can be at least partially ameliorated.

3.2 Qualitative features of the scattering cross section

A key feature of the scattering from an isolated plasmonic nanoparticle is the existence of a resonance peak in the scattering cross section. When neighboring particles are present, the single-particle resonance may be perturbed as a result of near-field interactions among particles. The resultant scattering cross section for the group of particles then manifests new polarization and angular complexity, as well as entirely new resonant features, compared to the scattering cross section for the isolated particles [30].

3.2.1 Definition of an aggregate

It is well known from Mie scattering theory that for the smallest particles only the dipole term contributes significantly to the scattering cross section. To evaluate qualitative scattering features it is appropriate to begin with consideration of this term and of the associated dipole-dipole coupling between closely-spaced particles. In the near-field, this coupling scales as $c_{\mathrm{nf}} = (r/R)^3$, where r is the particle radius and R is the distance between the particles. For higher multipole orders the

coupling scales as $c_{\rm nf} = x^{-l+1}(r/R)^{l+2}$, where l is the given multipole order and $x = kr$ is the radius of the particle in optical units. From energy conservation considerations, however, in the radiation zone coupling at any multipole order scales as $c_{\rm rz} = x^2(r/R)$. The transition between the two forms of coupling occurs for distances on the order of $R = 1/k$ with the near-field coupling dominating the radiation type for closer distances. In the discussion to follow it will be convenient to work in terms of the normalized inter-particle distance $a = R/r$. This normalization represents the observation that with respect to the dipole fields to a large degree aggregates with a similar ratio of particle radius to inter-particle spacing behave in a similar manner, with only a slowly varying dependence of these fields on the individual-particle radius as an isolated parameter.

For purposes of the present discussion, the term *aggregate* will be defined as a set constructed algorithmically as follows. The starting point for the construction is a set of particles with fixed, well-defined positions, and the end point is a *partition* of this set, that is, a sub-classification of this starting set into non-overlapping and non-empty subsets. Each of these non-overlapping subsets is defined as an *aggregate*. A subset is generated by starting with an arbitrary given particle not yet included in any previously generated subset. Surrounding particles similarly not yet included in any previously generated subset are included in the subset being generated, *singly and severally*[2], until further inclusion no longer affects the normalized partial scattering cross section of the given particle at the leading order of magnitude. Then, for each newly included particle, its surrounding particles are similarly evaluated for inclusion in the subset. This process is continued for the subset until no new particles are found for inclusion. The term *partial* cross section as used here

[2]Common cases exist where the simultaneous inclusion of a group of particles in the growing subset *will* affect the normalized partial cross section at leading order, but inclusion of the particles one at a time would not.

refers to the contribution to the total aggregate cross section due to the interaction of a given particle with the incident field, and in addition its interaction with the scattered fields of all other particles in the aggregate.[3]

There are many important problems for which the application of this algorithm results in the inclusion in a single subset of all particles in the starting set. For these cases the analysis of Section 3.3 below will still prove useful, but the scaling analysis of the present section is not relevant.

The magnitude of the Poynting vector for the scattered fields from any particle scales as $1/R^2$ in the far-field. For identical particles of radius r, the inclusion of any single particle within a radius of about $R = 3r$ approximates the above construction. Any particle beyond this radius produces a scattered Poynting vector magnitude at the given particle's surface that is less than $1/10$ that of the incident field. Using the normalized distance a, as introduced above, $R = 3r$ is seen to correspond to $a = 3$. This $a < 3$ limit is the widely used threshold for the *independent*-to-*multiple* scattering transition [2]. As all near-field effects decrease significantly more rapidly than $1/R^2$, this limit is also sufficient for the near-fields. Note that the discussion of Section 3.2.3 amends this argument slightly to treat correctly the case of more than one particle being present within a given radius, but otherwise the argument remains the same.

An overview of the general features of scattering from nanoparticle aggregates will be presented next. This will allow the identification of overall trends in the scattering behavior, which in turn will facilitate the development of a practical approximation technique.

[3] In the present case of coherent scattering, the magnitude of this partial cross section varies with the total number of particles in the aggregate and so it must be additionally normalized.

3.2.2 Widely-spaced particles; coherent structure factor

To a high degree of accuracy, groups of particles (or aggregates) with inter-particle spacing $a > 3$ may be treated as independent scatterers. However, since for the present biological labeling application the optical coherence length in many cases is significantly larger than $1/k$, which is the order of the imaging voxel size, the net effect of these independent scatterers involves a coherent superposition of individual fields. The result is that, if phase factors are neglected, N identical particles in an imaging voxel manifest a scattering cross section N^2 times that of the individual particle. An experimental situation, wherein this quadratic dependence of the wavelength-integrated scattering cross section has been conclusively demonstrated by the author and his colleagues, is discussed in detail in reference [65]. In the general case, however, the effect of the optical phase shift between individual scatterers must be included and the concept of the coherent *structure factor* is applicable. For a comprehensive treatment of this *structure factor* see references [67] and [68], and for its application in an experimental setting see the reviews by Timasheff and Townsend [69] and by Pritz and Lee et al. [70]. In addition to the contribution of more than one particle to the imaging voxel, there is also the possibility that more than one aggregate can contribute. However, for simplification of the present discussion it will be assumed that the optical modality used measures scattering from one aggregate at a time. In microscopic imaging applications this single-aggregate case is not rare, and it can easily be identified through the use of the integrated scattering intensity alone.

3.2.3 Closely-spaced particles

As discussed in Section 3.2.1, provided that the particles are small enough, the effects of inter-particle coupling can be evaluated by consideration of the dipole

component of the near-fields. In addition, when averaging over the angular orientation of an ensemble of aggregate structures, something that is done in many experimental measurement situations, the primary effect of inter-particle coupling is to induce a spectral shift to longer wavelengths in the resonance of the optical cross sections, that is, to induce a *red-shift*. (A more comprehensive analysis of this averaging procedure and its implications is presented in the author's Master's thesis [71], and in Kreibig[30].) These considerations lead to the treatment of the dipole coupling in terms of an effective *scalar* coupling and allow the effects of the angular symmetry of the dipole fields to be treated separately. Further, in this scalar-coupling approximation, the spectral position of the resonance of a specific particle is perturbed to varying degree depending predominantly on the number and position of the nearest neighboring particles. Therefore, the cross sections of aggregates with higher dimensionality of particle packing or with more closely-spaced packing will have larger spectral shift compared to the cross sections of aggregates with more open particle packing structure.

3.2.3.1 Total number of particles

The most important implications of the radial dependence of the coupling are that the degree of red-shift increases either when more particles are added to an aggregate structure or when the mean inter-particle distance within the structure is decreased (assuming that other aspects of the structure affecting red-shift remain unchanged). Despite the previously mentioned difficulties associated with the reconstruction of a specific aggregate structure corresponding to a cross-section measurement, in many experimental cases these simple rules allow a comparative approach to be taken.

For aggregates of closely-spaced particles, with respect to the total number of par-

ticles in the aggregate, saturation of spectral change occurs quickly. As illustrated in Figure 3.4 this fact can be demonstrated by considering the coupling effects of neighboring particles at gradually increasing distances from a given particle. This threshold region has a radius of $a = 10$ for planar aggregates and $a = 30$ for 3D aggregates. A simple calculation is used to arrive at these values. This calculation proceeds from the expression of an *effective* particle polarizability α_i for each particle "i" in the aggregate, expressed in terms of the polarizability of all of the isolated particles α_{ii}:

$$\alpha_i = \alpha_{ii} \left(1 + \sum_{j \neq i} c_{ij} \alpha_j \right). \tag{3.1}$$

Here α_i is the scalar *polarization* of the particle at $\vec{x}_i$, and α_{ii} is the intrinsic or isolated polarization. For purposes of discussion, details of the angular and polarization dependence have been suppressed. This calculation is further simplified through the treatment of the contribution from all particles in the aggregate as identical, that is, through the approximations $\alpha_i = \alpha$ and $\alpha_{ii} = \alpha_0$. For dipole-dipole coupling of a given mode, the coupling constant $c_{ij} \propto a_{ij}^{-3}$. For particles of an *areal* density ρ_a, the coupling contribution from a circular region of width dR at R is proportional to $(2\pi\rho_a/R^2)dR$. Given that $a = 2$ at closest approach, it is seen that at about $a = 10$ the coupling has been reduced by an order of magnitude from its value at closest approach. Analogous reasoning is used to derive the $a = 30$ threshold region for volume aggregates. In actual application, there are significant transition effects at the boundaries of these threshold regions and, in fact, each particle does not contribute equally to the aggregate polarizability. However, the calculation as presented does provide a useful leading-order estimate of these threshold distances.

In many biomedical applications symmetry effects are averaged out by the collecting optics, and in these cases the primary effect of the mean inter-particle distance

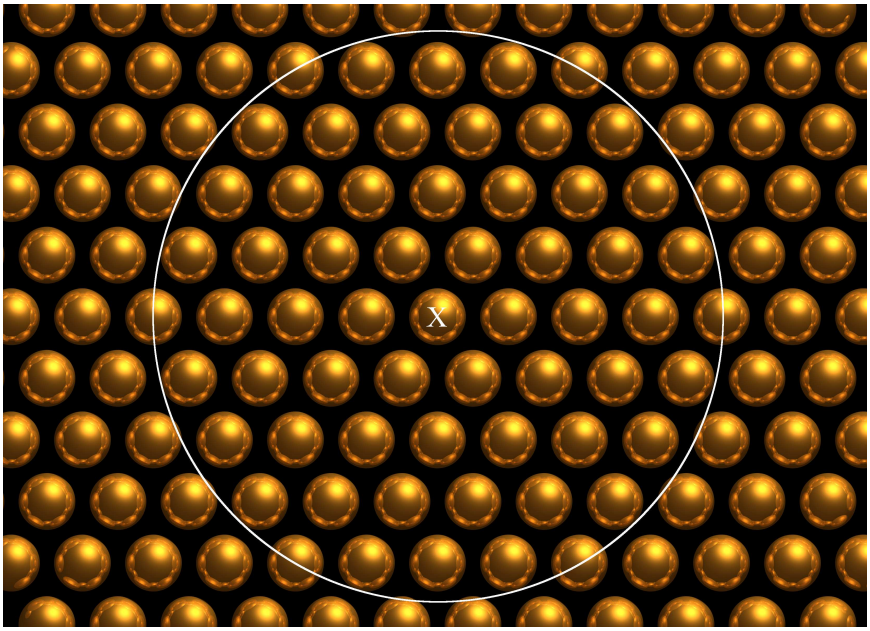


Figure 3.4: For the marked particle, the effects of near-field coupling from particles outside of the circular region can be neglected.

is on the red-shift of the wavelength of the maximum scattering cross section. Extremes of this shift are the single particle cross section at large inter-particle separation and the cross section of an infinite lattice for small separation in a large aggregate. Example plots of these spectra are shown in Figure 3.5. The spectral features of the cross section for an *infinite* periodic lattice can in principle be determined analytically by applying Floquet's (i.e., Bloch's) theorem [60]. For present purposes, however, these calculations have been implemented numerically[4].

[4]This particular simulation uses the Rayleigh approximation. A more accurate calculation using the coherent potential approximation (CPA) or using Waterman's formulas[72] would be straightforward but would not change the details of the discussion.

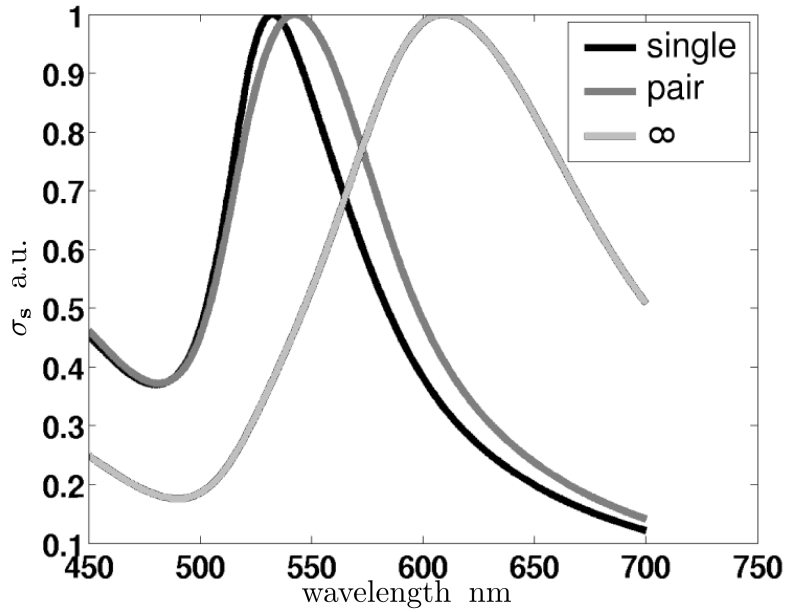


Figure 3.5: Scattering cross section for an Au particle, a pair, and an infinite hexagonal-close-packed (HCP) lattice with matching inter-particle spacing. Plots are normalized for comparison.

3.2.3.2 Aspect ratio

The asymmetry of an aggregate structure, expressed in terms of its *aspect ratio*, has two primary effects. The first is that it results in corresponding asymmetry in the magnitude of the optical cross sections for different incident-field polarizations (or, equivalently, for different aggregate orientations with respect to the incident field). For example, from consideration of the boundary surface circumscribing the entire aggregate, it is observed that the magnitude of scattering from an aggregate along a given sampling direction behaves as if the cross section were from a *hypothetical* spherical aggregate with a dimension corresponding to that of the *actual* aggregate's boundary surface along the given direction. This has been verified for single particles by numerical studies presented in Barber and Yeh [24] and, in

principle, because of the formal equivalence between the cluster-centered T-matrix and the single-particle T-matrix (see Chapter 4), these results must also apply to multiple-particle structures.

The second effect of the asymmetry of the aggregate is due to the angular symmetry of the electric dipole component of the near-fields[44]

$$\vec{E} = [3\hat{n}(\hat{n}\cdot\vec{p}) - \vec{p}]\,\frac{1}{r^3},$$

where $\hat{n}$ is the unit normal vector at the observation point and $\vec{p}$ is the polarization induced in response to the excitation fields. Inter-particle coupling producing spectral shift to longer wavelengths of the cross-section resonances is associated with the component of this dipole field aligned with the inter-particle axis. The angular dependence of this component is

$$E_{\parallel}(\theta) = 3\cos^2\theta - \cos\theta,$$

where θ is the zenith angle of the inter-particle axis, and the polarization of the exciting field is aligned with the polar axis.

It is evident that there will be a maximum zenith angle, for the alignment of the inter-particle axis with respect to the polarization axis, outside of which neighboring particles will not induce red-shift. Particles within the cone specified by this maximum angle will induce a red-shift in the cross sections, and particles outside of this cone will induce a blue-shift (or, equivalently, will cancel red-shift effects from other particles in the structure)[5]. This maximum zenith angle is

$$\begin{aligned} 3\cos^2\theta_c - \cos\theta_c &= 0 \\ \theta_c &= \cos^{-1} 1/3 \\ &\approx 70^\circ. \end{aligned}$$

[5] As the dipole-dipole coupling is proportional to $\cos\theta$ only $\theta \leq \pi/2$ need be considered.

The fact that the value of this angle is larger than nearest-neighbor angles in all close-packing arrangements for identical spheres adds to the generality of the Section 3.2.3.1 discussion.

In principle it would be possible to calculate an *optimal* aspect ratio that maximizes the spectral shift of the cross section. However, it appears that the value of this optimal aspect ratio will depend on the details of particle size and separation distances, because of additional geometric effects governing inter-particle coupling and because of the dependence of the induced polarization on particle radius. Both of these factors are outside the scope of the present discussion.

3.2.3.3 Particle packing

For aggregates containing a large number of particles, the details of the particle positions are most effectively described statistically through the use of a correlation function. This correlation function expresses the probability density for a neighboring particle to be at any position given the presence of a particle at the origin in the case of the pair-correlation function, or given the positions of more than one other particle for higher-order correlation functions. A qualitative discussion of the effects of the total number of particles, the mean inter-particle spacing, and the aspect ratio can be used to evaluate the effects of various forms of this correlation function.

With respect to the radial form of the correlation function, given a constant volume and provided that the angular part of the correlation function is uniform, the effect of a non-uniform *radial* correlation will always be to decrease the mean inter-particle distance compared to that associated with a statistically *uniform* correlation. This means that any *radial* non-uniformity is in general to be associated with increased inter-particle coupling.

When the angular part of the correlation function is considered separately for radial correlations that produce equivalent mean inter-particle distance, more chain-like angular correlation corresponds to increased inter-particle coupling while more plate-like correlation corresponds to decreased coupling.

3.3 Approximation formulas

The discussion of the qualitative aspects of the optical cross section for a nanoparticle aggregate supports the feasibility of using only a few independent variables for describing this complex system with its many degrees of freedom. The variables considered were N (the total number of particles in an aggregate), a (the normalized inter-particle spacing), x (the particle radius in optical units), and $\alpha_0(E)$ (the isolated-particle polarization as a function of energy). It remains to be shown how simple formulas useful for interpretation of experimental observations on a day-to-day basis might be constructed with these variables.

For this construction it is essential to make use of the fact that there are only slight differences in scattering cross section between aggregates differing only slightly with respect to inter-particle positions. To do this, in order to facilitate the approximation of the scattering cross section it is useful to express that cross section as the sum of two components. The first component is the scattering cross section from a *hypothetical* regular aggregate containing the same number of particles with the same $\langle \vec{x}_j \rangle$ as the *actual* aggregate. The second component accounts for the effect of random perturbations of the particle positions (see Figure 3.6).

The success of this approach depends on being able to evaluate the magnitude of the smaller term. It turns out that in many practical cases this second term can simply be neglected, that is, the cross section can be closely approximated by the

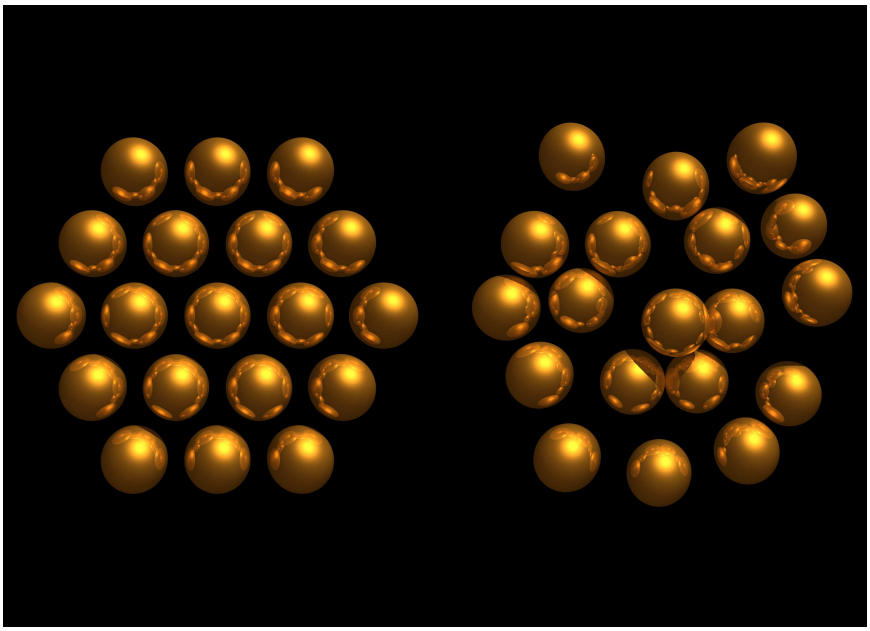


Figure 3.6: An ideal regular aggregate with particles at positions $\{\vec{x}_j\}$, and a randomized aggregate with the same $\{\langle\vec{x}_j\rangle\}$.

cross section of an ideal periodic aggregate with statistically equivalent particle positions. The factors affecting the suitability of this approximation are evaluated next. The derivation in this section is based in part on discussion presented in the series of articles by Tateiba [38, 39, 40, 41, 42]. These articles include an analysis of the range of physical systems to which the approach is applicable.

Formal operator notation will be used to describe the approach. The objectives are to provide a conceptual illustration and to assess magnitude of the perturbation term, but no attempt will be made to evaluate this term precisely. Because the expressions for the required operators are somewhat simpler, the *vector potential* $\vec{A}$ will be used instead of the electric field.

The equation for the total field is separated into deterministic and stochastic parts:

$$\begin{aligned}\langle\vec{A}\rangle &= \vec{A}_i + \sum_j \int_{\mathrm{R}^3} d^3x' \mathrm{G}^{(0)}(\vec{x},\vec{x}')\alpha(\vec{x}'-\vec{x}_j)\langle\vec{A}(\vec{x}')\rangle \\ &\quad + \left\langle \sum_j \int_{\mathrm{R}^3} d^3x' \mathrm{G}^{(0)}(\vec{x},\vec{x}')[\mathsf{U}(\delta x_j') - \mathsf{I}]^{-1}\alpha(\vec{x}'-\vec{x}_j)[\mathsf{U}(\delta x_j') - \mathsf{I}]\vec{A}(\vec{x}') \right\rangle\end{aligned}$$

where $\alpha(\vec{x}' - \vec{x}_j)$ contains all of the information about the particle positions and polarizability, U is the translation operator, I is the identity operator, and members of the set $\{\delta x_j\}$ are random perturbations for the set of particle positions $\{\vec{x}_j\}$.

This equation is expressed more compactly as

$$\langle\vec{A}\rangle = \vec{A}_i + \mathcal{G}^{(0)}\langle\vec{A}\rangle + \left\langle \mathcal{G}^{(1)}\vec{A} \right\rangle$$

with the random part contained completely in the third term.

To evaluate the effect of this random term, the *transverse* part of the free-space dyadic Green's function will be used [49]:

$$\overline{\overline{\mathcal{G}}}(\vec{x},\vec{x}') = \bar{\bar{\mathsf{I}}}\frac{\mu_0 e^{i|\vec{x}-\vec{x}'|}}{4\pi|\vec{x}-\vec{x}'|}. \tag{3.2}$$

Here $\bar{\bar{\mathsf{I}}}$ is the identity dyad and $\overline{\overline{\mathcal{G}}}(\vec{x},\vec{x}')\cdot\vec{j}$ is the vector potential $\vec{A}(\vec{x})$ of an oscillating current source $\vec{j}\delta(\vec{x}-\vec{x}')$.

This Green's function is applied in the integral, and the term retains information for each point source

$$\begin{aligned}\left\langle \mathcal{G}^{(1)}\vec{A} \right\rangle &= \left\langle \sum_j \left\{ \frac{\mu_0\alpha e^{i|\vec{x}-\vec{x}_j+\delta x_j|}}{4\pi|\vec{x}-\vec{x}_j+\delta x_j|}\vec{A}(\vec{x}_j+\delta x_j) \right.\right. \\ &\quad \left.\left. -\frac{\mu_0\alpha e^{i|\vec{x}-\vec{x}_j|}}{4\pi|\vec{x}-\vec{x}_j|}\vec{A}(\vec{x}_j) \right\} \right\rangle.\end{aligned} \tag{3.3}$$

For $\langle|\delta x|^2\rangle \ll 1$

$$\begin{aligned}\vec{A}(\vec{x}_j+\delta x_j) \approx & \vec{A}(\vec{x}_j) \\ |\vec{x}-\vec{x}_j+\delta x_j| \approx & |\vec{x}-\vec{x}_j|\left(1-\frac{\delta x_j\cos\gamma_j}{|\vec{x}-\vec{x}_j|}\right),\end{aligned}$$

where γ is the angle between $\vec{x}$ and $\vec{x}_j$. Equation 3.3 becomes

$$\begin{aligned}\left\langle \mathcal{G}^{(1)}\vec{A}\right\rangle &= \left\langle \vec{A}(\vec{x}_j)\right\rangle \frac{\mu_0\alpha e^{i|\vec{x}-\vec{x}_j|}}{4\pi|\vec{x}-\vec{x}_j|}\left\langle -i\delta x_j\cos\gamma_j\right\rangle \\ &= \mathcal{G}^{(0)}\langle\vec{A}\rangle\left\langle -i\delta x_j\cos\gamma_j\right\rangle.\end{aligned}$$

Note that even when the $\{\delta x_j\}$ have zero mean (i.e, $\langle\delta x_j\rangle = 0$), the $\langle -i\delta x_j\cos\gamma_j\rangle$ factor will be nonzero. The $\cos\gamma_j$ factor changes sign as δx_j does. This term has magnitude such that $|\langle\delta x_j\cos\gamma_j\rangle|^2 \leq \tilde{l}_j^2$ where $\tilde{l}_j$ is the correlation length of the δx_j. For many types of statistical distribution, the correlation length is independent of the index j and is of the same order of magnitude as the inter-particle spacing [37, 73]. The implication is that, for inter-particle spacing $\vec{x}_{jl} = k\vec{r}_{jl} \ll 1$, the effect of the random term can be completely neglected.

In cases where this term does need to be calculated, an iterative approach may be used [38]:

$$\begin{aligned}\langle\vec{A}\rangle_0 &= \vec{A}_i + \mathcal{G}^{(0)}\langle\vec{A}\rangle_0 \\ &\Rightarrow \\ \langle\vec{A}\rangle_n &= \vec{A}_i + \mathcal{G}^{(0)}\langle\vec{A}\rangle_n + \left\langle \mathcal{G}^{(1)}\langle\vec{A}\rangle_{n-1}\right\rangle.\end{aligned}$$

Each iteration adds a factor of $\alpha\tilde{l}^2$, and the series converges rapidly when $\tilde{l} \ll 1$.

3.3.1 A numerical experiment

Given the applicability of the approximations of the previous discussion, it remains to be demonstrated that simple approximation formulas can actually be constructed.

Ideally such formulas would be derived from analytical expressions which include inter-particle coupling effects limited to the first few multipole orders. Waterman has derived and presented such expressions for particle assemblies on *infinite* periodic lattices of various types[72]. Although it would not be difficult to limit his expressions to particle assemblies of finite extent, the complexity of his expressions precludes widespread usefulness of such approximation formulas, with the possible exception of using them to guide the implementation of numerical calculation. With this in mind a simpler approach will be taken here, an approach that uses numerically calculated cross sections from a series of ideal aggregates to demonstrate that a few-parameter formula can actually be achieved in experimentally relevant situations.

In order to produce a specific formula for experimental application, it is necessary first to specify the range of variation for each of the *effective* variables defining the aggregates. Then a series of numerical simulations is performed, using aggregates specified on ideal lattice positions (generally using a hexagonal-close-packed (HCP) lattice) and with each of the effective variables changing in a gradual manner. The full matrix of simulation results is then used to produce optimized polynomial approximations for any desired experimental observable, such as scattering peak position or scattering peak width.

Here an approximation will be developed in terms of the complex-modulus of the aggregate polarization $\alpha(E)$ as the fundamental quantity for approximation. All of the various cross sections can be expressed in terms of this polarization. In particular, the scattering and extinction cross sections are

$$\sigma_s = \frac{8\pi^4}{\lambda^4}|\alpha|^2, \text{ and} \tag{3.4}$$

$$\sigma_e = 4\pi \Im\mathfrak{s}\alpha, \tag{3.5}$$

where λ is the incident free-space wavelength. Additional independent variables expressing incident and scattered angle and polarization dependency are suppressed, with the specifics of these variables depending on experimental requirements.

Using simple analysis of the interaction of the dipole component of near-fields, it is possible to predict the gross effects of the effective variables, and one expects the overall polarization to scale roughly as

$$\alpha(E) = N\alpha_0(E)\left\{1 - ga^{-3}\alpha_0(E)\right\}^{-1}, \tag{3.6}$$

where a *geometric* factor g is used to include the averaged effects of the number and relative positions of the neighboring particles within the aggregate. Note that even this simple scaling formula can include the effect of the scattering-peak dependence on N, if the parameter g (which can be expressed as a function of N) is taken as an *effective* parameter. In fact, using a simulation matrix to obtain approximations for the parameters in this formula is a potential approach to an approximation of the aggregate polarization $\alpha(E)$ although the approach is so simplified that it would likely be of quite limited practical use. A more general approach is suggested in the following.

To provide a specific example and to illustrate several important fine points, a partial approximation will be constructed for HCP planar aggregates of an intermediate, 19-particle size. No attempt will be made yet to construct a full multidimensional approximation, although it should be kept in mind that that is the ultimate

objective and does not in principle increase difficulty except for computation-time requirements. As illustrated in Figure 3.7, a graded series with gradually decreasing inter-particle spacing shows both the expected red-shift and broadening of the main scattering peak but, in addition, the series shows the gradual prevalence of a complex multiple-peak structure [6].

To make the best use of these computationally-expensive calculations, each vector of cross-section results is fitted to multiple asymmetric Lorentzian functions, where incident energy (or, equivalently, wave-number) is used as the parameter. The specific form of asymmetric Lorentzian function used is

$$A(E; E_0, \gamma, a_0, a_1) = a_0 \frac{\gamma}{\pi\left((E-E_0)^2+\gamma^2\right)} - 2a_1 \frac{\gamma(E-E_0)}{\pi\left((E-E_0)^2+\gamma^2\right)^2}, \quad (3.7)$$

which function is parametrized in terms of the location E_0, the peak half-width at half-maximum γ, the normalized amplitude a_0, and the asymmetry parameter a_1.

This response function is appropriate for a singly-resonant system with dispersion, and for the current application it has been found numerically to be superior to other fit functions, such as the more commonly used "linear" (i.e., non-derivative form) asymmetric Lorentzian function, the asymmetric Gaussian function, or a more general Fano-resonance function. The actual curve fitting is implemented using the Levinburg-Marquardt method[74]. It has been ascertained that, even in the case of the most complicated spectra shown in Figure 3.7, the use of two resonances is sufficient to allow matching of the wavelength-integrated cross section to within 0.1%.

[6] For details of the numerical calculation methods used in this section see Chapters 4 and 5.

When the location of the shortest wavelength peak is extracted from the graded simulation series of Figure 3.7, it is well-approximated by a third-degree polynomial as shown in Figure 3.8. The fact that a third-degree polynomial is sufficient to approximate the peak position is consistent with the preceding discussion of how the inter-particle coupling behaves when the individual-particle radii and spacing are sufficiently small that only the dipole-dipole term is significant (see Equation 3.1 and the discussion following it). It is particularly interesting to see how well this approximation technique works even in the presence of the more complicated peak structure that occurs at closer inter-particle spacing.

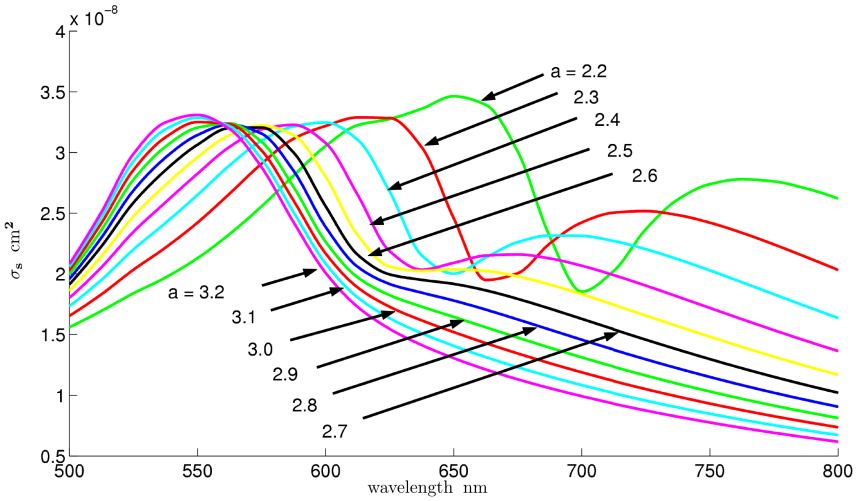


Figure 3.7: Scattering cross sections for 19-particle, planar HCP aggregates with gradually decreasing inter-particle spacing $a = 3.2$ to $a = 2.2$ (units of particle radius r).

At this point, using suitably-selected simulation series, a full multidimensional approximation would normally be constructed for peak-position as a function of both a and N, although this will not be demonstrated here. The details of this simulation matrix depend critically on experimental specifics, and they relate to those specifics

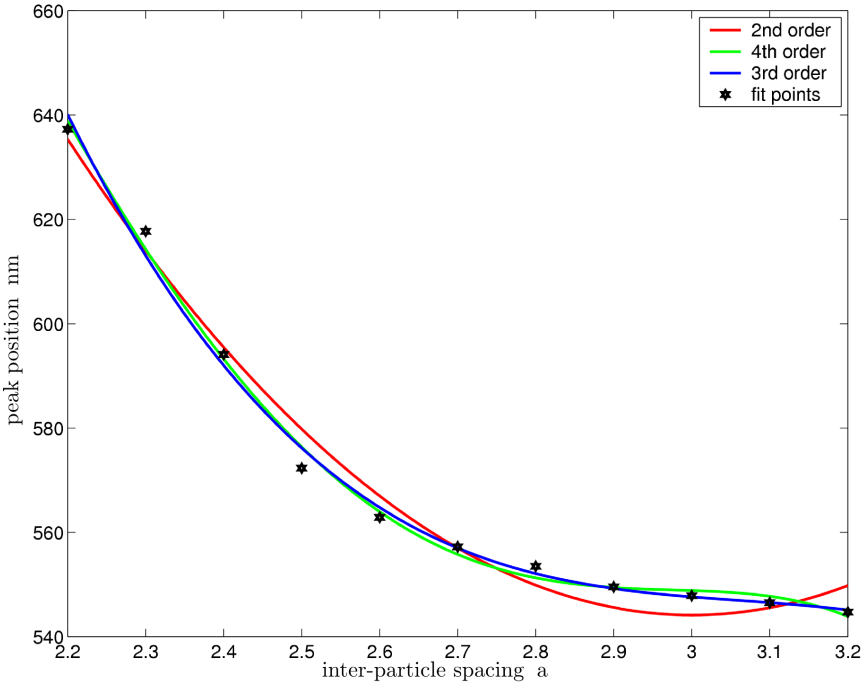


Figure 3.8: Wavelength scattering maxima corresponding to Figure 3.7 and polynomial approximations of varying degree.

in another important way. Toward the end of avoiding complex cross-section features if at all possible, they can be used to help guide the use of nanoparticle aggregates as effective labels for biophysical applications. It is expected that a matrix of 100 simulation points will allow the construction of this approximation with an integrated cross-section accuracy of near 1%, an accuracy sufficient for all of the experimental applications motivating the present analysis.

3.4 Summary

In the discussion of the present chapter a particular phenomenological approach for the prediction and description of scattering cross sections from biologically relevant

nanoparticle aggregates has been elaborated. The approach involves developing a qualitative, physics-based interpretation of the features of the aggregate cross section and then applying this interpretation to developing few-parameter formulas for approximating the scattering spectra. The parameters used in these formulas can be instantiated by fitting to judiciously-selected computational results, or even to results from experimental observations. The approach has been exemplified by construction of a partial approximation that represents, for a specific planar aggregate with a fixed number of particles, the dependence on inter-particle spacing of the spectral maximum of the scattering cross section.

The value of any phenomenological approach such as this, and of the resulting approximations, is largely determined by their practical usefulness, that is, by the degree to which they can be used to predict or to encapsulate experimental observations. Because of the complexity of this specific scattering system, different approximations will be optimal for different experimental applications. It is hoped that the discussion has illustrated how this approach can be adapted or extended as required to cover alternative or additional features of the scattering cross section, such as features associated with aggregate aspect ratio, or with angular symmetry and optical polarization. This would be accomplished by identifying suitable effective variables and constructing for them approximations derived either analytically, or from the reduction of a matrix either of simulation-based or of experimentally-derived results.

Chapter 4

Multiple-particle systems

4.1 Introduction

Before continuing with development of the phenomenology (in Chapter 6), it will be necessary to extend the theoretical formalism developed in Chapter 2 so that it covers the full multiple-particle case. This extension involves the construction of the multiple-particle T-matrix from the T-matrices of the component particles. Because each of these component T-matrices is expressed with respect to a VSW basis referred to a distinct origin, a translation operator for the VSW basis is required, and working through the details of this operator is the most difficult part of the construction. The end result is a *multi-centered* T-matrix represented on a *redundant* VSW basis.

The most widely-used application of the T-matrix technique is in the analysis of electrodynamic cross sections[75], and for this application the T-matrix form generally used is the *cluster-centered* or *superposition* form. This re-expresses the multi-centered T-matrix in terms of a single VSW basis referred to the common origin of the multiple-particle system. More recent applications of the T-matrix technique are in the analysis of per-particle and sub-structure contributions to the electrodynamics of the system, as well as in the calculation of the full electrodynamic fields[ibid.]. For these purposes the full multi-centered form must be retained.

In addition to being a theoretical formalism, the T-matrix technique is also a com-

putational technique for solving electrodynamics problems. Computational techniques like this are broadly classified into techniques that solve Maxwell's equations expressed in differential form and those that solve the equations expressed in integral form. The latter techniques may require additional constraints on the system such as regions with piecewise-homogeneous constitutive relations. Further classification of techniques is based on whether or not low-order (or, equivalently, "compact") or high-order basis function representations are used for the fields, and on whether or not these basis functions express the fields directly or indirectly as vector or scalar potentials for the fields. From the perspective of this classification, it is seen that the T-matrix technique uses a *high-order* direct basis for the fields and, implicitly in its formulation in terms of the continuity of the tangential components of the fields across boundary surfaces, solves Maxwell's equations expressed in *integral* form.

4.2 T-matrix as an abstraction

Any electrodynamic system may be effectively analyzed in terms of the change that the system causes to the fields in response to an excitation field:

$$\vec{E} = \vec{E}_i + \vec{E}_s. \tag{4.1}$$

Here the excitation and response fields are denoted as the *incident* and *scattered* fields, respectively. Further, in a completely general manner a transfer operator provides a mapping between the incident field and the scattered field:

$$\vec{E}_s = T\vec{E}_i; \tag{4.2}$$

This transfer-operator is referred to as a *T-matrix*[1][23].

[1]This formulation does not require the T-matrix to be independent of the incident field but, in further discussion here, it will be assumed that it is independent, that is, that the system responds

This approach in terms of field changes can be extended further, leading to approximations like the *order-of-scattering* approach[76] or the *distorted-wave Born* approximation[77]. The particular multi-particle T-matrix technique elaborated here represents a slightly different approach but, in a certain sense, it must also be seen as an approximation. However, it defers any approximation onto the definition of the material from which the system and its surrounding medium is composed, onto the concept of the surfaces bounding the scattering particles, and onto the physical accuracy of the associated boundary condition expressions. Given applicability of these material and boundary-surface constraints, the technique is then *exact* in its *analytical* formulation. It thus becomes possible in any practical computational implementation of the technique to calculate the electrodynamic properties of any system to an accuracy limited only by computational resources.

Using more indirect approaches, computational electrodynamic techniques other than the particular analytic approach discussed in this chapter could be used to calculate the T-matrix. In fact, the approximations mentioned above, as well as almost any technique applicable to computational electrodynamics, could be applied to this end, with at least some degree of effectiveness. However, for some of these techniques conversion of calculation results to a T-matrix representation would be less than straightforward.

4.3 Translation operator for the VSW basis

The translation operator for the VSW basis is formally defined by the relation[78]

$$\vec{M}_{lm}^{(1)}(kr,\hat{n}) = A_{lm}^{l'm'}(kR,\delta\hat{n})\vec{M}_{l'm'}^{(1)}(kr',\hat{n}') + B_{lm}^{l'm'}(kR,\delta\hat{n})\vec{N}_{l'm'}^{(1)}(kr',\hat{n}'), \quad (4.3)$$

linearly to the excitation field.

where A and B are the *vector* addition coefficients, $\vec{M}_{lm}$ and $\vec{N}_{lm}$ are the transverse VSW basis functions, $R = |(r\hat{n} - r'\hat{n}')|$, and $\delta\hat{n} = (r\hat{n} - r'\hat{n}')/R$. In Relation 4.3 magnetic VSW $\vec{M}$ may be exchanged with electric VSW $\vec{N}$ without modifying the relation and, in addition, the relation may be generalized, as required, to transformations between *outgoing* and *regular*fields or between *outgoing* and *outgoing* fields.

The particular form of this operator used in any specific application depends on two considerations. First, it depends on the boundary conditions affecting the VSW basis itself, that is, with the transformation from either of *regular* or *outgoing* fields to either of *regular* or *outgoing* fields. Second, it depends on the specifics of the translation *regions*. These latter regions are the interior and exterior of a spherical boundary surface of radius equal to the radial magnitude of the translation and are illustrated conceptually in Figure 4.1.

Following the notation of Mackowski and Mishchenko [79], it is convenient to denote the outgoing-wave and regular-wave translation operators by the letters associated with the spherical Bessel function component of the associated VSW:

$$H_{l,m}^{l',m'}(kr,\hat{n}) \quad \text{outgoing-wave translation operator,}$$
$$J_{l,m}^{l',m'}(kr,\hat{n}) \quad \text{regular-wave translation operator.}$$

Note that in application the $J_{l,m}^{l',m'}$ form of the translation operator never modifies the regularity of the associated VSW, whereas the $H_{l,m}^{l',m'}$ form is applied to outgoing VSW and produces regular VSW[2].

[2]For special applications the development of additional forms of translation operators for more general boundary conditions, such as those involving *incoming-wave* VSW, would be straightforward.

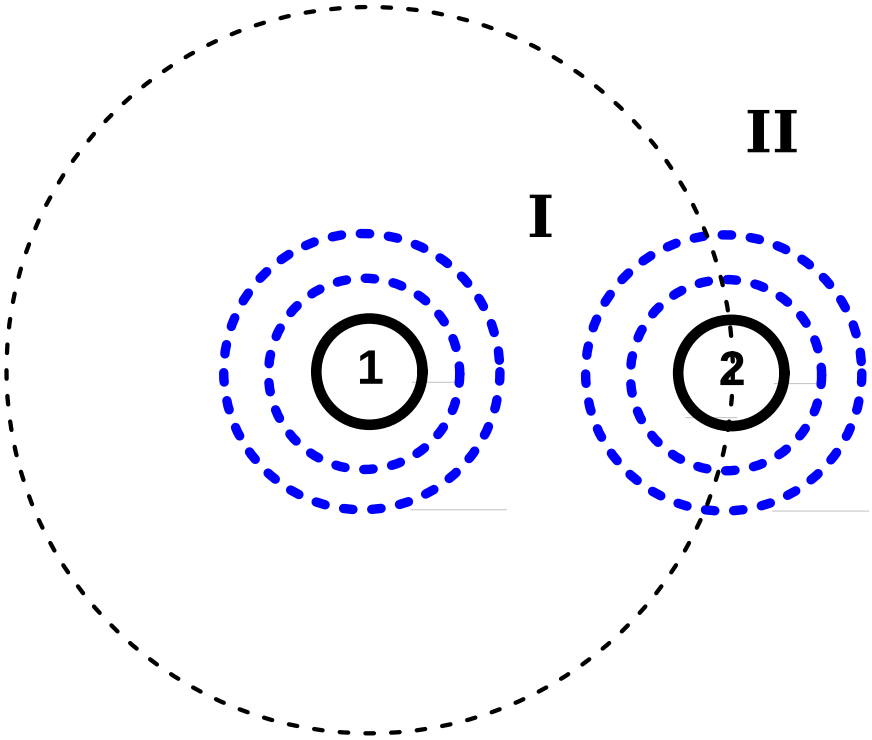


Figure 4.1: Diagram of boundary surface governing application of the VSW translation operator for translation from origin “2” to origin “1”.

Further specifics governing application of the translation operator are [ibid.]

Region	destination B.C.	source B.C.	operator form
I	*regular*	*regular*	J
I	*regular*	*outgoing*	H
II	*regular*	*regular*	J
II	*outgoing*	*outgoing*	J

The next sections address the closed-form expressions and the recurrence relations for the coefficients representing the translation operator but, because of the length and complexity of the associated derivations, only an overview will be presented

here.

4.3.1 Closed-form expressions

The vector addition coefficients were originally expressed in closed form by Stein [80] and Cruzan[81], using a lengthy derivation based on the *addition* theorems for the spherical harmonics and for the spherical Bessel functions. Closed-form expressions useful for practical computation are due to Borghese [34] and these are used in the numerical codes supporting the present work (see Section 5.3.2). By application of the translation operator to its *eigenfunctions* (which are plane-waves $\vec{E}_0 \exp(i\vec{k} \cdot \vec{r})$) and by direct comparison with the recurrence-relation-based implementation of the same operator, these complicated expressions have been shown to be precisely correct. Unfortunately, the requirements for several additional special coefficients (in particular, the *Gaunt* functions and the *Clebsch-Gordan* coefficients) limit the usefulness of these closed-form expressions in practical application.

4.3.2 Recurrence relations

In contrast to the use of closed-form expressions, the use of recurrence relations provides a straightforward technique for calculating the vector addition coefficients. Usable recurrence relations were derived by Mackowski [78] and by Chew[82], both of whom derive expressions for stable recurrence relations suitable for calculation of the scalar and the vector addition coefficients. The discussion of the present section follows the derivation presented in Mackowski [78] but with some notation changes.

The derivation of the vector addition coefficients generally involves the calculation

of the scalar addition coefficients, which translate the *scalar* spherical harmonics as

$$u_l(kr)Y_{lm}(\hat{n}) = C_{lm}^{l'm'}(kR, \delta\hat{n})v_{l'}(kr')Y_{l'm'}(\hat{n}') \tag{4.4}$$

where u_l and v_l are spherical Bessel functions appropriate to the specific boundary conditions affecting the fields, and other notation is as for Equation 4.3.

Because the gradient operator itself is translation invariant it can be applied to both sides of Equation 4.4 with the result being a new equation that is treated component-wise by working separately with each *helical* basis[3] component. An application of Equation 4.4 itself allows the LHS and RHS of these component equations to refer to scalar harmonics referred to the same origin, which may then be canceled. And, finally, the recurrence relations for the Legendre and Bessel functions are used to simplify these expressions and thereby construct relations for the addition coefficients themselves. The construction of recurrence relations for the vector addition coefficients in terms of these relations for the scalar coefficients is then straightforward.

Mackowski points out that the translation operator can be factored, that is, represented in terms of a composition of rotations and a linear translation along the polar axis. As the representation of each of these constituent operations is diagonal with respect to the multipole order and the azimuthal degree[4], respectively, this enables a reduction in computational complexity at the point when the translation operator is actually applied. For example, in application to a matrix, the non-factored representation requires $\mathcal{O}(N^3)$ operations compared with the factored representation requiring only $\mathcal{O}(N^2)$ operations, where N is a fixed expression involving the

[3]The *helical* basis is an irreducible representation isometric to the $l = 1$ angular momentum space, which representation is constructed in a straightforward manner from the standard Cartesian basis vectors[51].

[4]In the present work, the standard "degree" and "order" nomenclature of the *spherical harmonics* is *reversed* in order to be consistent with colloquial usage.

maximum multipole order. Factoring thus clearly enables a significant speed-up of translation-operator application, something frequently done during the T-matrix calculations.

For the numerical computations supporting the research described here, codes based on the closed-form expression [34] and on the more recent recurrence-relation based method [78] have both been completed and validated. This application of two very different techniques to calculate the translation-operator coefficients and the successful matching at machine precision of outputs from the separate techniques provides an important accuracy check for the representation of this very complex operator.

4.3.3 Limitations of matrix representations

An unfortunate consequence of the exponential radial dependence of the VSW basis functions of multipole order l when $kr \ll l$ is that matrix representations of the H form of the translation operator may be highly ill-conditioned[83]. The condition number of these matrices increases *exponentially* with the maximum multipole order corresponding to the radius of the spherical surface circumscribing the multiple-particle system. Issues related to this are discussed in more detail in Section 4.4.3.

4.4 Multiple-particle T-matrix

The derivation of the present section follows that of Borghese[34] but with different notation. The definition of the VSW basis, along with the associated translation operator, allows the relations from Equations 4.1 and 4.2 to be expressed explicitly.

The incident and scattered fields are defined by vectors of coefficients for the regular

and outgoing VSW, respectively:

$$\vec{E}_i = \sum_p \sum_{lm} E^{(p)}_{i:lm} \tag{4.5}$$

$$\vec{E}_s = \sum_p \sum_{lm} E^{(p)}_{s:lm}, \tag{4.6}$$

where

$$\begin{aligned}\vec{E}^{(p)}_{i:lm} =& a^{(p)}_{i:lm}\vec{M}^{(1)}_{lm} + b^{(p)}_{i:lm}\vec{N}^{(1)}_{lm} \\ \vec{E}^{(p)}_{s:lm} =& a^{(p)}_{s:lm}\vec{M}^{(3)}_{lm} + b^{(p)}_{s:lm}\vec{N}^{(3)}_{lm},\end{aligned}$$

and the spherical tensor indices "l" and "m" denote the multipole order and azimuthal degree, and "i" and "s" denote the incident and scattered fields, respectively. The parenthesized super-script (p) denotes a specific VSW basis centered at the origins of particle "p", and the super-scripts (1) and (3) denote *regular* and *outgoing* fields, respectively[5]. Equations 4.5 and 4.6 describe the fields as a *superposition* of contributions from multiple *redundant* VSW bases, each centered at the origin of a specified particle.

The T-matrix for the multi-particle system is then defined by the relation

$$E^{(p)}_{s:lm} = T^{(p,q):l'm'}_{lm} E^{(q)}_{i:l'm'}. \tag{4.7}$$

Expressions such as those in Equation 4.7 involve contraction between the contravariant and co-variant *tensor* indices, as well as summation over repeated indices in the super-scripts[6], following the Einstein convention. In all cases of application

[5] From this point on additional indices referring to the *magnetic* $\vec{M}_{lm}$ and *electric* $\vec{N}$ VSW will be suppressed.

[6] An exception to this convention occurs when an index also appears on the left-hand side of an equation.

in the following, the only *matrix* operators will be the transfer-operator T and the translation operators H and J, and field vectors are always E[7]. This allows suppression of all the spherical tensor indices without ambiguity. Further, the *regularity* class of the VSW can always be deduced from context, as the transfer operators in most cases of application always transform from *regular* to *outgoing* fields, the J-form of the translation operator does not modify the regularity of the fields, and the H-form of the translation operator transforms from *outgoing* to *regular* fields[8].

The key concept involved in the derivation of expressions for the coefficients of the multi-centered T-matrix defined in Equation 4.7 is that of the *effective* incident field at a given particle, and this is illustrated conceptually in Figure 4.2. This effective field includes the actual incident field at the specified particle and, in addition, the superposition of the scattered fields from all other particles in the system.

Then, the expression for the multi-centered scattered fields of Equation 4.7 can be expressed alternatively as

$$E_s^{(p)} = T^{(\underline{p})} E_{\text{eff}}^{(p)}, \text{ where} \tag{4.8}$$

$$E_{\text{eff}}^{(p)} = E_i^{(p)} + \sum_{q \neq p} H^{(p,q)} T^{(q,r)} E_i^{(r)}, \tag{4.9}$$

and the notation $(\underline{p})$ is used to indicate the *single-particle* T-matrix for particle "p". At this point, Equations 4.7 and 4.8 in combination implicitly define the coefficients of the multi-centered T-matrix

$$\left[\delta^{(p,q)} - \left(1 - \delta^{(p,q)}\right) T^{(\underline{p})} H^{(p,q)}\right] T^{(p,q)} = \delta^{(p,q)} T^{(\underline{p})}. \tag{4.10}$$

[7] E is generally used to indicate a field vector, although most expressions will apply without modification when the magnetic field H is used instead.

[8] It is to be understood implicitly that these transformations are to be generalized as appropriate to other boundary conditions, for example, when expressing T-matrices and translation operators suitable for operation on fields within the *interior* of a structure.

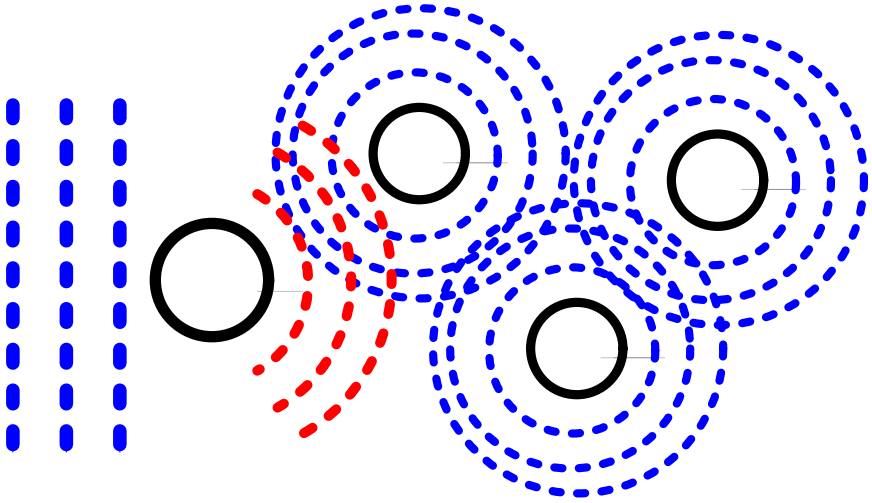


Figure 4.2: Schematic representation of the *effective* incident field at a given particle.

Given suitable truncation conditions for matrix representations of the operators and fields concerned, the problem has been reduced to linear algebra. With respect to these truncation conditions, the Debye criterion (see section 2.2.2.1) allows the calculation of a suitable maximum multipole-order corresponding to any required accuracy in the calculation of the scattered fields and, with slight modification, in the calculation of the near-fields[9].

4.4.1 Redundant multipole basis

The use of multiple VSW bases in Equations 4.5, 4.6, and 4.7 requires further comment. If extended to arbitrary multipole order, any *single* VSW basis is closed and complete with respect to representation of fields satisfying specified boundary conditions[49]. In this case the use of multiple VSW bases would be redundant.

[9] In this regard, see the discussion of Section 6.4.1.

However, when *truncated* with respect to multipole order, a VSW basis can only represent fields to a certain degree of accuracy. No angular details corresponding to higher multipole orders can be represented, and the representation accuracy will be strongly localized to a spherical region about the origin. The implications of this are that, when truncated VSW bases referred to distinct origins are combined, both the accuracy of the fields representation and the size of the region wherein this accuracy can be maintained increase. Unfortunately, this increase in accuracy of representation comes with several disadvantages. First, the combined fields basis is no longer constructed from orthogonal functions. Second, the ability of the combined basis to support spatial detail with respect to fields representation changes from location to location and depends in a complicated way on the specifics of the particle positions. In application, the lack of orthogonality of the basis functions does increase computational complexity. However, the fact that the VSW basis functions themselves are of such high order compared to basis functions used by alternative computational techniques means that this only causes a small inconvenience. And, further, any difficulties associated with a lack of spatial uniformity in the accuracy of the field representation are ameliorated by the fact that in most cases these spatial details are associated with the individual particles themselves and thus with the regions where accuracy of representation is greatest.

4.4.2 Cluster-centered approach

During the calculation of the optical cross sections, the multi-centered incident and scattered fields of Equation 4.7 are re-expressed in terms of a single VSW basis referred to a common origin, usually taken as the cluster center. A tremendous simplification occurs with respect to both storage requirements and computational time, if this translation operation is performed in advance. This is accomplished by cen-

tering and accumulating the dual-centered T-matrices into one *cluster-centered*[10] T-matrix, as[79]

$$T^{(\underline{c},\underline{c})} = J^{(\underline{c},p)} T^{(p,q)} H^{(q,\underline{c})} \tag{4.11}$$

where "$\underline{c}$" denotes the common origin, and the super-script "($\underline{c}$,$\underline{c}$)" indicates the *cluster-centered* T-matrix.

Applying the analysis of Waterman[23], it is evident that this cluster-centered T-matrix is formally equivalent to a single-particle T-matrix for an arbitrary structure contained within a circumscribing spherical surface, as illustrated conceptually in Figure 4.3. An implication of this is that, in order to preserve the accuracy of the calculation, the maximum multipole orders used for the T-matrices of the particles comprising the structure and those used for the cluster-centered T-matrix itself will in general be different, with the latter requiring a higher maximum multipole order[79].

4.4.2.1 Mackowski-Mishchenko formulation

An elegant implementation of the multiple-particle T-matrix computation is the algorithm of Mackowski and Mishchenko[79], which algorithm is available as a FORTRAN code in the public domain and thus is widely used. This algorithm uses the fact that the solution of the system of Equation 4.10 and the centering operation of Equation 4.11 can proceed a *column* at a time and the fact that the extinction cross section depends on the trace of the T-matrix (see Equation 2.42). In combination these two facts allow the algorithm to solve the system to gradually increasing multipole order and to terminate the calculation when the desired accuracy has been

[10] In the literature, this is also known as the *superposition* T-matrix.

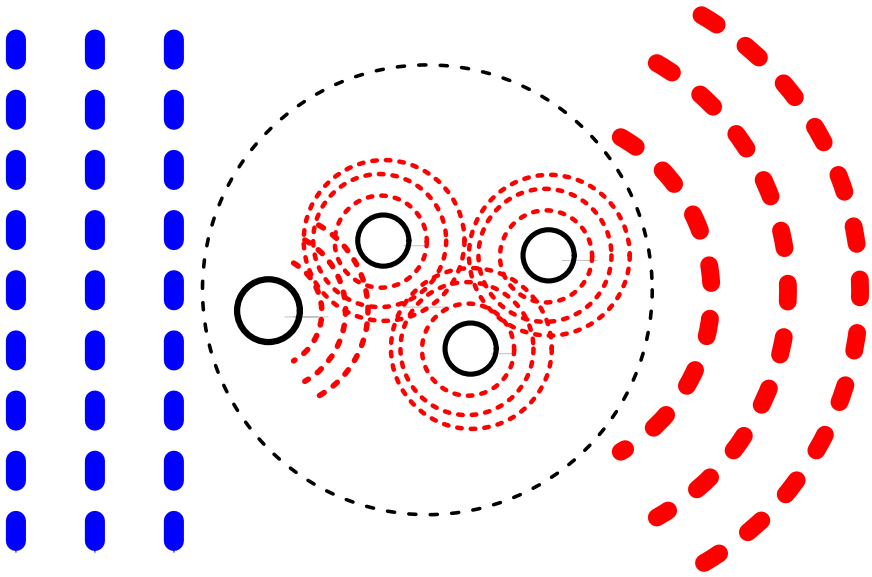


Figure 4.3: Representation of the circumscribing spherical boundary surface used to define the *cluster-centered* T-matrix.

achieved, where that desired accuracy is referred to the extinction cross section. This convergence-control property of the algorithm is applied not only with respect to collective-multipole-order contributions to the extinction cross section but also with respect to per-particle contributions. Further, given that the only required storage is for the cluster-centered T-matrix itself and that the translation operators can be calculated at the time of use, the algorithm is especially appropriate for calculations involving very large numbers of particles.

4.4.3 Multi-centered approach

4.4.3.1 Full fields calculation

There is a subtle distinction between the expressions for the incident and scattered fields appearing in Equations 4.7 and 4.9. In particular, the *effective* incident field expression of Equation 4.9 requires the *total* incident field expressed in the VSW

basis of a specific particle and, for this reason, Equation 4.7 also requires this *total* incident field. On the other hand, the expression of the scattered fields represents the superposition of the contributions from all of the single-centered fields. Apart from this detail all difficulties relating to the calculation of the full electrodynamic fields are computational and are discussed in Section 5.3.6.

4.4.3.2 Partial cross sections

The discussion of the present section is based on Stout[84] but with different notation. In analogy to what was done in Section 2.2.3, the time-averaged Poynting vector for the total fields can be separated into components contributing to the absorption, the scattering, and the extinction cross sections, respectively. From Equation 4.7 these expressions are

$$\begin{aligned}
\langle S_i \rangle &= \frac{1}{2}\Re\left(\vec{E}_i \times \vec{H}_i\right) \quad \text{incident flux} \\
\langle S_s \rangle &= \sum_{p,q} \frac{1}{2}\Re\left(\vec{E}_s^{(p)} \times \vec{H}_s^{(q)}\right) \quad \text{scattered flux} \\
\langle S_e \rangle &= \sum_{p} \frac{1}{2}\Re\left(\vec{E}_i \times \vec{H}_s^{(p)} + \vec{E}_s^{(p)} \times \vec{H}_i\right) \quad \text{extinction flux.}
\end{aligned}$$

In these expressions the per-particle contributions to the flux can be separately identified

$$\begin{aligned}
\langle S_s^{(p)} \rangle &= \sum_{q} \frac{1}{2}\Re\left(\vec{E}_s^{(p)} \times \vec{H}_s^{(q)}\right) \quad \text{per-particle scattered flux} \\
\langle S_e^{(p)} \rangle &= \frac{1}{2}\Re\left(\vec{E}_i \times \vec{H}_s^{(p)} + \vec{E}_s^{(p)} \times \vec{H}_i\right) \quad \text{per-particle extinction flux.}
\end{aligned}$$

This leads to the concept of the *partial* cross section. Such cross sections represent power transfer between the individual particles and the collective fields associated with the entire multiple-particle structure. Implications of the fact that this power

transfer can go in *both* directions are that partial scattering cross sections can be negative and that the energy conservation relation of the complete cross sections, viz., $\sigma_e = \sigma_a + \sigma_s$, does not constrain the partial cross sections.

For reference, the partial scattering cross section is

$$\sigma_s^{(p)}\left(\theta_i, \phi_i : \hat{\beta}_i)\right) = \frac{1}{k^2} \sum_{q,r,s} \Re\left(E_i^\dagger J^{(\underline{c},s)} T^{(p,s)\dagger} J^{(p,r)} T^{(r,q)} J^{(q,\underline{c})} E_i\right),$$

where "$\underline{c}$" indicates translation with respect to the cluster center and E_i represents the vector of coefficients for the *normalized* incident field in direction (θ_i, ϕ_i) with polarization $\hat{\beta}_i$.

The corresponding partial extinction cross section is

$$\sigma_e^{(p)}\left(\theta_i, \phi_i : \hat{\beta}_i)\right) = -\frac{1}{k^2} \sum_{q} \Re\left(E_i^\dagger J^{(\underline{c},p)} T^{(p,q)} J^{(q,\underline{c})} E_i\right).$$

Analogous expressions can be constructed for the other kinds of cross section.

4.4.3.3 Recursive formulation

The fact that the matrix blocks of the system of equations represented by Equation 4.10 are *infinite* raises the problem that the *solution of a truncated* infinite system of equations will in general not be the same as the *truncation of the solution* of the same system. To see this it is sufficient to consider matrix products representing the composition of *irregular* and *regular* translation operators as in

$$\left[H^{(p,r)}\right]_{l_{\max}} \left[J^{(r,q)}\right]_{l_{\max}} \neq \left[H^{(p,r)} J^{(r,q)}\right]_{l_{\max}} \tag{4.12}$$

where the sub-scripted square brackets indicate truncation with respect to the maximum multipole-order of the VSW bases used in the representation of the translation operators. For this particular problem, it is these products that cause the most difficulty because of the ill-conditioned nature of the matrix representations of the

irregular form of the translation operator[84, 85]. These particular products will occur explicitly in any factorization process solving Equation 4.10 directly and will occur implicitly in any iterative solution process.

One solution to this problem, the one used in the standard algorithm for the cluster-centered T-matrix calculation, is simply to increase the truncation multipole-order sufficiently that the inaccuracy in these products doesn't unduly affect the accuracy of the solution. The difficulties associated with calculating at this high multipole-order are somewhat ameliorated by the optimized convergence-control features built into the algorithm, features that allow the *circumscribing* multipole-order corresponding to truncation to be selected on a per-particle basis and to be gradually increased only to a maximum that produces a solution of the required accuracy[79].

An attractive alternative approach that avoids many of these difficulties is to express Equation 4.10 in a form that makes use of the fact that a composition of translation operators can be formally expressed as a single translation, that is, as

$$H^{(p,r)}J^{(r,q)} = H^{(p,q)}. \tag{4.13}$$

A recursive solution can then be developed that builds up the multiple-particle system one particle at a time and that explicitly makes use of Relation 4.13 for all products involving the irregular translation operators. The approach described here is essentially that proposed by Stout[83].

An equation additional to Equation 4.10 is required for the recursion step. This is provided by expressing the scattered fields from a particle in the (N-1)-particle cluster in terms of the *effective* incident field (see the discussion at Equation 4.9) of

the complete cluster (that is, of the N-particle cluster) thusly:

$$E_s^{(p)} = T_{N-1}^{(p,q)} E_{\text{eff}}^{(q)}, \text{ where} \tag{4.14}$$

$$E_{\text{eff}}^{(q)} = E_i^{(q)} + \sum_{r \neq q} H^{(q,r)} T_N^{(r,s)} J^{(s,q)} E_i^{(q)} \tag{4.15}$$

and where the subscripts $N-1$ and N indicate the dual-centered T-matrices of the (N-1)-particle and the N-particle systems, respectively. Equation 4.14 is sufficient to enable the definition of additional relations defining $T_N^{(p,q)}$ depending on its position in the block matrix structure. For the specifics of these additional relations see Equations 33a,33b,35a, and 35b of Stout [ibid.].

The first step of the recursion process is obtained by inspection from Equation 4.10 and builds the solution for the two-particle system from the single-particle T-matrices of the first two particles (see Equations 28 of Stout [ibid.]).

Notwithstanding the formal advantages of this recursive approach, there are several major practical disadvantages. The most significant is that, due to the fact that at each step of the recursion the solution is for a *distinct* multiple-particle system, during the solution process there is no straightforward manner with which to estimate the accuracy of the solution so far achieved. Other disadvantages of the approach relate to problems associated with the efficiency of its computational implementation for systems containing large numbers of particles. These problems are discussed in more detail in Section 5.3.4.

4.5 Summary

The discussion of the present chapter extended the T-matrix formalism developed in earlier chapters for the single-particle case to the treatment of multiple-particle systems. This extension constructed the multiple-particle T-matrix out of the single-

particle T-matrices of the component particles. This construction is expressed in terms of multiple *redundant* VSW bases, each referred to the origin of one of the component particles.

This use of multiple bases necessitated the application of a translation operator for the VSW, and an overview was given of the derivation and application of this operator. It was pointed out that the most effective approach for its calculation uses recurrence relations and that, when the operator is computationally applied, computational expense is greatly reduced by factoring the operation into a combination of rotations and a single linear translation aligned with the polar axis. It was further pointed out that, due to unavoidable consequences of the radial dependence of the VSW basis functions, matrix representations of the *irregular* form of the operator are highly ill-conditioned, with the specifics depending on the multipole-order corresponding to the spherical surface circumscribing the multiple-particle system.

For application to the analysis and calculation of optical cross sections, the multiple-particle T-matrix is simplified and re-expressed in terms of a single VSW basis referred to a common origin. This cluster-centered T-matrix (or, alternatively, superposition T-matrix) then facilitates high-speed calculation of cross sections and enables evaluation of collective electrodynamics of the complete multiple-particle system. A top-level description of the algorithm most widely used for the computation of this cluster-centered T-matrix was presented. The algorithm is especially noteworthy in that, in order to achieve efficient convergence to the desired tolerance, it optimizes on a per-particle basis the maximum multipole-order used during the calculation.

For application to full-fields calculation or to analysis of electrodynamic behavior associated with the individual particles making up the system, the full multiple-particle T-matrix must be used. A unique analysis provided by this form of the T-

matrix evaluates the transfer of energy flux, as expressed in the per-particle *partial* cross sections, between the individual particles and the collective system.

Finally, a *recursive* alternative algorithm that might enable more accurate calculation of the multiple-particle T-matrix in certain situations was discussed and critiqued. This recursive formulation formally combines compositions involving the *irregular* translation operator into a single operator application. However, it was concluded that, before it can be put to practical use, the algorithm requires further refinement with respect to estimation of solution accuracy and with respect to handling large numbers of particles.

Chapter 5

Computational methods

5.1 Overview

At the outset of numerical code-base development, the design focus was on building a set of tools enabling application, to electrodynamics problems, of computational methods of a higher order and more analytic nature than those that have previously been commonly used. A less ambitious focus on more specific requirements would have resulted in a less powerful set of tools but would have entailed significantly less development investment. The objectives of two different research projects determined choice of the first alternative. The first, with Jochen Guck's research group at the University of Leipzig, involved the study of optical interactions with biological cells and tissue, where it will be observed that, at near-infrared wavelengths, cells typically have a size of $kr \approx 100$ optical units, and can be modeled effectively with a few tens of objects. The second project, with Konstantin Sokolov's research group at the University of Texas at Austin, involved the study of optical interactions with nanoparticle aggregates, where aggregates typically have a size of $kr \approx 1$ optical unit but may contain many thousands of particles. The requirements of these two particle-size and particle-number domains with respect to implementation of the multiple-particle T-matrix algorithm, though not in opposition, are quite different. In particular, evaluation of the electrodynamics of relatively large, asymmetric objects requires development of an arbitrary-precision implementation, which can often make use of sparse matrix structures, while evaluation of smaller structures

containing a large number of particles requires development of a fully-scalable parallel implementation, which generally uses only dense matrix structures.

5.1.1 Design goals

The desire to apply multiple-particle T-matrix calculations to the problem domains of both the Texas and the Leipzig groups imposed on the code-base implementation the following requirements:

- *arbitrary-precision* arithmetic capability, due to considerations of matrix condition number associated with T-matrix calculations at high multipole order;
- *hybrid-parallel* design, that is, a design utilizing both thread-level and process-level parallelism, due to trends in CPU core number and the need to handle T-matrix calculations involving a large number of particles;
- *object-oriented* design, given modern programming "best practices" regarding code maintainability, optimization of the code feature-development cycle, and code ease-of-use.

It was decided that, although simultaneously satisfying this combination of requirements would be difficult, doing so was both possible and feasible. The success of the code-base development has demonstrated that, not only was the decision correct, but it has resulted, first in a quite general code-base applicable to almost all problems for which the multiple-particle T-matrix approach is appropriate and, second, in a code-base enabling people working on such problems to more fully exploit the power of recently introduced and imminent advances in computer software and hardware technologies, for example, advances involving parallel processing.

5.1.2 Dependencies

Wherever possible, existing libraries and algorithms are used in the numerical code-base. Primarily, the open-source GNU compiler tool-chain[86] is used, although the entire code-base has also been successfully compiled with the INTEL[87] and the PGI[88] C++ compilers. The mere arbitrary-precision library is directly based on the C-language GNU multi-precision library (GMP)[89] and the C-language multi-precision-floating-point-real library (mpfr)[90]. The requirement for this new C++ library is explained in more detail in section 5.2.1. Linear algebra classes and algorithms are implemented using the fully-templated generic matrix methods gmm[91] library, with extensions as discussed in section 5.2.3. Multi-process, distributed linear algebra and iterative linear algebra solvers are implemented using the TRILINOS library from Sandia National Laboratories[92]. Fast Fourier transforms of real and complex numbers, at arbitrary precision, are implemented using the FFTW library[93], with minor extensions to allow the use of this library with templated number types. Transforms onto the 2-Sphere (i.e., as discrete Legendre transforms) are implemented using the S2KIT library[94]. It is notable that apart from the above, all of the special function implementations, although utilizing primitive functions from the C++ run-time library and based on discussions in the literature, are herein completely new. The requirement for this is explained more fully in section 5.2.2.

5.2 Key features

5.2.1 Arbitrary-precision number classes

Initially the code utilized existing arbitrary-precision number libraries implemented for C++, first the NTL[95] and then the CLN[96] libraries, but dependence on these

libraries had to be phased out. The `NTL` dependency was phased out because of `NTL`'s lack of a complete primitive function implementation, especially with respect to the $\tan^{-1}$ function. The `CLN` dependency was phased out because of `CLN`'s unusability for multi-threaded application. `CLN`'s reference-counter-based garbage-collection implementation precludes thread-safe operation. Modification of the `CLN` library was briefly considered but conversion of the library to support thread safety did not appear to be straightforward. For these reasons a new C++ arbitrary-precision number library `mere` was implemented. This new library provides all features required for arbitrary-precision calculation and serves as a minimalist, robust C++ interface to the C-language libraries `GMP`[89] and `mpfr`[90].

5.2.2 Numerical functor class

The set of primitive functions provided by the C++ run-time library and additionally provided by the arbitrary-precision class used in the present code-base (see section 1.6.7) generally includes functions such as `sqrt` and `pow`, transcendental functions such as `exp` and `log`, and the trigonometric functions and their inverses. For each of these functions, the number of parameters is small and behavior of function evaluation can proceed without the library's needing to provide a user with choice among multiple variants of this behavior. This does occasionally cause problems as when, for example, the library-specified value of the expression $0.0^{0.0}$ as determined by evaluation of the function call `pow(0.0, 0.0)` changes from one release of the run-time library to the next, but in practice such problems are sufficiently rare that they can be tolerated.

For the more complicated functions used in mathematical physics the number of possible parameters is often much larger, the expected behavior of function evaluation often has multiple variants and possibly even conflicting definitions, and the

computational cost of function evaluation can be large. To address these problems the computer-programming concept of *functor* (sometimes referred to with "function object" rather than "functor") is useful. A functor allows separating the computational processing associated with a function into distinct phases, an *initialization* phase that prepares the functor's *state*, and an *evaluation* phase that engages the functor's *methods* to perform a function evaluation and generate a result. A functor's state includes such things as values for less-commonly-modified parameters, information affecting normalization and behavior of special cases, and the results of expensive initialization done once (that is, the first time the results are needed) and then saved, thus enabling initialization cost to be amortized over many function evaluations.

A functor is defined so that in simple cases a user can ignore all these details, and defaults will determine how function evaluation proceeds. The defaults are defined to enable the evaluation to be as general as possible. In more common cases, normalization details need to be specified prior to function evaluation. Taking as an example the associated Legendre functions $P_{l,m}$, there is the classical normalization where $P_{l,-m} = (-1)^m \frac{(l-m)!}{(l+m)!} P_{l,m}$ and the more modern normalization $P_{l,-m} = (-1)^m P_{l,m}$ used when the function appears as part of a spherical harmonic[97], along with several possible normalizations for the derivative that allow function evaluation to return otherwise singular results without numerical overflow. Finally, for special cases such as the evaluation of the Gaunt coefficients $G \begin{matrix} l_1 & l_2 & l_3 \\ m_1 & m_2 & m_3 \end{matrix}$ at arbitrary precision for high multipole order (l_1, l_2, l_3) the computation of these coefficients is so expensive the only practical way to allow repeated access to them in a computational model is to pre-calculate all values that may possibly be required (during the initialization phases of early uses of the functor), to have tables holding the results of this pre-calculation persist as parts of the

functor, in a database resident on hard disk, and to have a later use of the functor obtain the values it needs not with re-calculation but with table look-up.

This *functor* concept is realized in the code-base through the numericalFunctor base class (see Section 1.7.98). Functors for particular mathematical functions are then derived from this base class. The implementation of the `numericalFunctor` class is based in part on the functor implementation approach used in the excellent open-source `LTI-Lib` image-processing library[98], which the author used extensively in a previous project.

For any particular numerical functor (that is, any instance of the `numericalFunctor` class), a typical evaluation sequence proceeds as follows:

1. The functor's state is initialized by instantiating the `numericalFunctor::parameters` sub-class, setting the attributes as required, and passing this structure to the functor's `setParameters` method. Note that this initialization step can possibly prepare for *multiple* evaluation steps. In the case where the `setParameters` method returns `false`, the functor's `getStatusString` method will indicate what particular parameter attribute or initialization problem caused the return.

2. The function is evaluated using any one of the functor's `apply` methods. Generally `apply` methods return `true` if an evaluation is successful and `false` otherwise. The evaluation result itself is almost always passed as a *reference* parameter to the `apply` method.

3. If the `apply` method returned `false`, the functor's `getStatusString` method will indicate what caused the evaluation to fail, and appropriate action can be taken.

Depending on the particular numerical functor, what actually occurs during the `setParameters` activation of the initialization step can be more or less complicated. For example, the legendreFunctor (see Sections 1.7.93 and 5.2.2.2) merely initializes a vector of normalization factors. On the other hand, for the gauntFunctor (see Sections 1.7.88 and 5.2.2.5), when the use of a look-up table is requested and the required table has never been generated, all of the required table values will be calculated and inserted into a new table. Note that such complicated behavior is never the default behavior of a functor and is activated only with the explicit request of the user through modification of the appropriate boolean flag in the parameters structure.

A question might be raised as to why the code-base includes so many new implementations of *standard* mathematical special functions. It does so because *arbitrary-precision* implementations of these functions were not readily available when the code-base was being developed. As discussed in section 5.4, the situation in this respect may have changed quite recently thus possibly enabling the core evaluation code of many of these implementations to be replaced by calls to widely-used libraries offering significantly more validation and maintenance support than the author has means to provide.

Several of the more important numerical functor implementations are discussed in the remainder of this section.

5.2.2.1 Spherical Bessel functions

Because of their direct relationship to the trigonometric functions, *spherical* Bessel functions are actually easier to calculate than Bessel functions of integral order. For

example, starting with the definitions

$$j_0(x) = \frac{\sin(x)}{x}, \qquad j_1(x) = \frac{\sin(x)}{x^2} + y_0(x),$$
$$y_0(x) = -\frac{\cos(x)}{x}, \qquad y_1(x) = -\frac{\cos(x)}{x^2} - j_0(x),$$

and the recurrence relation

$$z_{l-1}(x) + z_{l+1}(x) = \frac{2l}{x} z_l(x),$$

where z_l is either of j_l or y_l, j_l and y_l can be calculated to arbitrary order l [97]. However, despite this special case for spherical Bessel functions, it was desirable to implement for Bessel functions a calculation method that would be generally applicable at arbitrary real order. With this goal in mind, the calculation approach used is based on the application of five steps:

1. a continued-fraction expansion is used to calculate the starting values at order $l = l_{\max}$ for the regular function and its derivative;

2. a decreasing-order recurrence relation is applied to generate all required orders to $l = l_{\min}$;

3. an additional continued-fraction expansion is used to calculate the irregular function and its derivative at $l = l_{\min}$;

4. the Wronskian relation is used to normalize the values at $l = l_{\min}$; and finally

5. an increasing-order recurrence relation is used to calculate the irregular function and its derivative to order $l = l_{\max}$[99].

During the increasing-order recurrence of the last step, it is convenient to normalize the full set of values of the regular function and its derivative. This approach

has the additional benefit that it avoids an implicit dependence on the library implementation of the trigonometric functions, which is especially important when a calculation is done with a complex value for the argument. The `sbesselFunctor` class is based on this approach and, as a fully-templated implementation for calculating the spherical Bessel functions, provides simultaneous calculation of j_l and y_l (or optionally $h_l^{(1)}$ and $h_l^{(2)}$) and their derivatives.

5.2.2.2 Legendre functions and transforms

The very first problems for which this code-base was developed involved the calculation of the full electrodynamic fields of large dielectric objects with azimuthal symmetry. Generally, the incident field used in these cases was a plane wave, or some other field such as as a Gaussian beam with low order, and thus contained only a few values of the azimuthal order[1]. As a result the corresponding T-matrix calculations involved all values of the multipole degree l up to some maximum value, but only the same few values of the azimuthal order m. For these reasons the desired implementation of the Legendre functions $P_{l,m}$ applies recurrence relations with respect to the multipole degree to calculate all required multipole degree values l, at a fixed value of the azimuthal order m.

Starting with the relations

$$\begin{aligned} P_{l,-m}(x) =& \frac{2^{-l}\left(1-x^2\right)^{\frac{l}{2}}}{l!}, \text{ and} \\ P_{l,m}(x) =& 0: \quad l < |m|, \end{aligned}$$

[1] In mathematical nomenclature the terms "order" and "degree" are applied to the indices of the associated Legendre functions in this manner. This is in *reverse* of the colloquial usage in respect to the VSW basis which refers to multipole *order* and not to multipole *degree*.

and the recurrence relation

$$(l-m+1)P_{l+1,m}(x) = (2l+1)P_{l,m}(x) - (l+m)P_{l-1,m},$$

$P_{l,m}$ can be calculated for values of $l \in [|m|, l_{\max}]$[97]. The only additional considerations involve the correct treatment of several special cases which affect the value of the derivative of $P_{l,m}$. Specifically, when $|m| = 1$ and $|x| \approx 1$, the limiting case must be treated specially in order to avoid numerical overflow.

Using an alternative *spherical tensor* representation of the VSW basis (see Reference [51]), the surface integrals required for the single-particle T-matrix calculation can be efficiently calculated through use of the discrete Legendre transform:

$$\tilde{f}_{l,m} = \sum_{k=0}^{2B-1} P_{l,m}(x_k)\, f(x_k) \text{ where,}$$
$$x_k = \cos\left(\frac{\pi(k+1/2)}{2B}\right),$$

and $f(x)$ is any function with suitably limited bandwidth.

It is interesting that, although direct formulations for this transform completely analogous to those of the widely-used fast Fourier transform (FFT) are possible, these formulations tend to be numerically unstable. However, an efficient indirect formulation based on the FFT algorithm itself is possible[94].

The legendreFunctor class (see Section 1.7.93) is based on these considerations and provides a fully-templated implementation for calculating the associated Legendre functions and their derivatives as well as the forward and inverse discrete Legendre transforms.

5.2.2.3 Jacobi functions

The rotation operator for the spherical harmonics $Y_{l,m}$ is expressed as

$$D^j_{\mu m}(\alpha,\beta,\gamma) = \exp(i\mu\gamma) d^j_{\mu m} \exp(im\alpha),$$

where (α,β,γ) are the Euler rotation angles, and the $d^j_{\mu m}$ are elements of the Wigner rotation matrix. These matrix elements can be further expressed in terms of Jacobi polynomials $P_n^{(\mu,\nu)}(x)$ of order (μ,ν) and degree n as[100]

$$d^j_{\mu m}(\beta) = \left[\frac{(j-m)!(j+m)!}{(j+\mu)!(j-\mu)!}\right] \left(\cos\frac{\beta}{2}\right)^{m+\mu} \left(\sin\frac{\beta}{2}\right)^{m-\mu} P_{j-m}^{(m-\mu,m+\mu)}(\cos\beta).$$

The rotation operator for the VSW basis is identical to that for the $Y_{l,m}$.

Starting with the relations

$$P_0^{(\mu,\nu)}(x) = 1 \quad \text{and} \quad P_0^{(\mu,\nu)'}(x) = 0,$$

where the prime indicates differentiation with respect to the argument x, and the recurrence relations

$$\begin{aligned}
&2(n+1)(n+\mu+\nu+1)(2n+\mu+\nu)P_{n+1}^{(\mu,\nu)} = \\
&\qquad (2n+\mu+\nu+1)\left[(2n+\mu+\nu)(2n+\mu+\nu+2)x + \mu^2 - \nu^2\right] P_n^{(\mu,\nu)} \\
&\qquad - 2(n+\mu)(n+\nu)(2n+\mu+\nu+2)P_{n-1}^{(\mu,\nu)} \\
&\qquad \text{and,} \\
&(2n+\mu+\nu)\left(1-x^2\right) P_n^{(\mu,\nu)'} = \\
&\qquad n\left[(\mu-\nu) - (2n+\mu+\nu)x\right] P_n^{(\mu,\nu)} \\
&\qquad + 2(n+\mu)(n+\nu)P_{n-1}^{(\mu,\nu)},
\end{aligned}$$

$P_n^{(\mu,\nu)}(x)$ and $P_n^{(\mu,\nu)'}(x)$ can be calculated for values of $n \in [0, n_{\max}]$[97]. The only special cases involve $|x| \approx 1$ for the derivative, which must be treated explicitly, and

particular values of μ and ν for which the starting point for the recurrence relations must be initialized carefully in order to avoid attempted division by zero.

The jacobiFunctor class (see Section 1.7.91) provides a fully-templated implementation for calculating the Jacobi polynomials and their derivatives.

5.2.2.4 Gaussian-Lobotto quadrature

Numerical integration is an important part of single-particle T-matrix calculation when applied to objects with arbitrarily-shaped boundary surfaces. A possible technique to accomplish this integration uses discrete Legendre transforms, and this is briefly discussed in Section 5.2.2.2. For validation and for application at arbitrary precision to functions not directly expressed in terms of associated Legendre functions, an additional integration technique is required.

Because they are evaluated by applying recurrence relations, essentially all of the special functions evaluated as part of the T-matrix calculation produce simultaneous output for all values of one index, at a specific value of the argument. Therefore, in common with the Legendre-transform approach, any acceptable numerical integration technique must enable utilization of these multiple values and should not simply discard values not presently required. Many of these values may eventually be needed in the course of the calculation, in particular, when the calculation for a future integration cycle requires precisely the same values of the argument. To satisfy this desideratum numerical integration was implemented using the Gaussian-Lobotto adaptive-quadrature[101] method enhanced with a cache structure capable of storing information associated with each specific quadrature point. The primary advantage of the Gaussian-Lobotto quadrature is that, as the quadrature points are adapted to finer and finer argument scales in response to the convergence requirements of the integration, each successive scale makes use of the quadrature points

from the preceding scale[102].

To enable use of this quadrature caching scheme when integrating an arbitrary function, a special signature is used for the calling parameters of the function, a signature that passes the cache node itself as one of the parameters. If a cache node so passed has already been initialized, this means that it contains information previously stored by the same function during a previous evaluation at exactly the same value of the argument. For example, if the integration sequence is passing from lower to higher orders of a discrete index for some required special-function evaluation, the cached information would generally contain the previously calculated special-function evaluations at all orders up to some maximum index value.

In practice, the degree to which this quadrature-caching scheme allows calculation results to be re-used depends on the relationship of the integrand of one integration cycle to the integrand of the next. For T-matrix calculations specifically, at least when relatively smooth boundary surfaces are involved, the efficiency of calculation-result re-use, expressed as a percentage of how many calculations would be required if no re-use were possible, often approaches 99%. Although the present implementation of the scheme requires that a special type of numerical functor be implemented for each specific integrand, an extension that would allow the integrand to be specified at run-time is feasible.

The gaussLobattoQuadratureFunctor class (see Section 1.7.90) provides a fully-templated implementation of numerical integration using adaptive Gaussian-Lobotto quadrature and employing quadrature caching as just described. To integrate an arbitrary function the continuumFunctor class (see Section 1.7.73) must be specialized. An example of such specialization, one used for single-particle T-matrix calculation for particles with azimuthally-symmetric boundary surfaces, is the azimuthalIntegrandFunctor class (see Section 1.7.70).

5.2.2.5 Gaunt, Wigner-$3J$, and Clebsch-Gordan coefficients

Functions with discrete argument values, like coefficients with sets of integer subscript indices, allow for special treatment. Where calculation of these coefficients is computation intensive, as is often the case when calculating at high numerical precision, an effective technique calculates in advance all coefficients likely to be required and then, at the point a particular one is needed, simply does a table look-up.

The Gaunt coefficients are defined as [103]

$$G \begin{matrix} l_1 & l_2 & l_3 \\ m_1 & m_2 & m_3 \end{matrix} = \int_\Omega Y_{l_1 m_1}(\hat{n}) Y_{l_2 m_2}(\hat{n}) Y_{l_3 m_3}(\hat{n}) d\hat{n} \tag{5.1}$$

$$= 0: \quad \text{any } |m_i| \qquad > l_i$$

$$= 0: \quad m_1 + m_2 + m_3 \qquad \neq 0 \tag{5.2}$$

$$= 0: \quad l_1 + l_2 + l_3 \mod 2 \qquad \neq 0. \tag{5.3}$$

Ignoring the Relations 5.2, and 5.3, the number of distinct coefficients corresponding to a specified value of the maximum multipole order $l_{\max}$ would be $\mathcal{O}\{l_{\max}^6\}$. However, taking into account these relations as well as additional symmetry considerations, the required number of stored distinct *non-zero* coefficients is only $\mathcal{O}\{l_{\max}^5\}$. As noted by Rasch [103], an efficient scheme for storage of these coefficients can be based on converting each index sextuple $\begin{pmatrix} l_1 & l_2 & l_3 \\ m_1 & m_2 & m_3 \end{pmatrix}$ to a *standard form* $\begin{pmatrix} l'_1 & l'_2 & l'_3 \\ m'_1 & m'_2 & m'_3 \end{pmatrix}$ where $l'_1 \geq l'_2 \geq l'_3$, $m'_1 \geq 0$, and $m'_3 = -(m'_1 + m'_2)$ and then performing a simple transformation to relate the value of the required coefficient to that of the corresponding standard-form coefficient.

With some additional improvements this storage scheme is used in the gauntFunctor class (see Section 1.7.88), which provides a fully-templated implementation based

on table look-up. As the Gaunt coefficients are also directly related to the Wigner-3J and Clebsch-Gordan coefficients[103]

$$\begin{pmatrix} l_1 & l_2 & l_3 \\ m_1 & m_2 & m_3 \end{pmatrix} = G \begin{matrix} l_1 & l_2 & l_3 \\ m_1 & m_2 & m_3 \end{matrix} \left\{ \sqrt{\frac{(2l_1+1)(2l_2+1)(2l_3+1)}{4\pi}} \begin{pmatrix} l_1 & l_2 & l_3 \\ 0 & 0 & 0 \end{pmatrix} \right\}^{-1},$$

and

$$\langle l_1 m_1 l_2 m_2 | \, l_3 m_3 \rangle = (-1)^{l_1 - l_2 + m_3} \sqrt{2l_3+1} \begin{pmatrix} l_1 & l_2 & l_3 \\ m_1 & m_2 & -m_3 \end{pmatrix}, \text{ respectively,}$$

these additional coefficients are provided by the wigner3JFunctor class (see Section 1.7.121), which is derived from the `gauntFunctor` class with only minor modifications mainly pertaining to behavior of the "even-l sum rule" (Relation 5.3).

Because of the high-order dependence of the total number of coefficients on multipole order $l_{\max}$, when working at large values of $l_{\max}$ it is necessary at any one time to load only look-up tables corresponding to a subset of the coefficients. To enable this the multipoleSextuple class (see Section 1.7.96) provides a set of methods supporting iteration over, and index arithmetic on, sextuples where optionally the azimuthal order m spans a *sparse* set of values.

The Gaunt coefficient values have been successfully verified to a very high multipole order ($l_{\max} = 1000$). This was done simply by comparing the product of two arbitrary functions $f(x)$ and $g(x)$ (generally taken as random functions) calculated in two different ways. The first way used the discrete Legendre transforms of the functions (see Section 5.2.2.2)[2] combined with the appropriate Gaunt coefficients.

[2]For these tests, the values of m_1, m_2 and m_3 are fixed.

The second way used direct multiplication, that is

$$f(x) \cdot g(x) = \sum_{(l_1,l_2,l_3)} P_{l_1,m_1}(x) G \begin{matrix} l_1 & l_2 & l_3 \\ m_1 & m_2 & m_3 \end{matrix} \tilde{f}_{l_2,m_2} \tilde{g}_{l_3,m_3}.$$

5.2.2.6 Vector Spherical Harmonics

A formulation for the VSW basis functions especially convenient for cross-section calculations is in terms of the Vector Spherical Harmonics (VSH) $B_{lm}(\hat{n})$, $C_{lm}(\hat{n})$, and $P_{lm}(\hat{n})$[49]:

$$\vec{M}_{lm} = z_l(x)\vec{C}_{lm}(\hat{n}) \tag{5.4}$$

$$\vec{N}_{lm} = \sqrt{l(l+1)}\frac{z_l(x)}{x}\vec{P}_{lm}(\hat{n}) + \frac{(xz_l(x))'}{x}\vec{B}_{lm}(\hat{n}) \tag{5.5}$$

$$\vec{L}_{lm} = z_l'(x)\vec{P}_{lm}(\hat{n}) + \sqrt{l(l+1)}\frac{z_l(x)}{x}\vec{B}_{lm}(\hat{n}), \tag{5.6}$$

where the radial functions $z_l(x)$ are selected from the set of spherical Bessel and Hankel functions $\{j_l(x), n_l(x), h_l^{(1)}(x), h_l^{(2)}(x) \ldots\}$ as appropriate to the boundary conditions, and the VSH themselves are defined as

$$\begin{aligned} \vec{P}_{lm} &= \hat{r}Y_{lm}(\hat{n}), \\ \vec{B}_{lm} &= \frac{1}{\sqrt{l(l+1)}} r\nabla\left(Y_{lm}(\hat{n})\right) \\ &= \hat{r} \times \vec{C}_{lm}, \text{ and} \\ \vec{C}_{lm} &= -\frac{1}{\sqrt{l(l+1)}}\vec{r} \times \nabla\left(Y_{lm}(\hat{n})\right). \end{aligned}$$

These VSH are especially convenient for cross-section calculations, as C_{lm} and B_{lm} express the limiting behavior of the angular part of the transverse VSW in the *far-field*. For this reason when evaluation of the VSW is required for either a cross-section or a full-fields calculation, the Formulations 5.4, and 5.5 are used.

The vectorSphericalHarmonicFunctor class (see Section 1.7.119) provides a fully-templated calculation of the VSH and is used during cross-section and full-field evaluations by the classes representing the single-particle T-matrix and the associated vector field ket (see Sections 1.7.168 and 1.7.170, respectively).

5.2.3 Linear algebra classes

As discussed in Section 5.1.2, for linear algebra calculations with vectors and 2D-matrices the code-base uses the `gmm` template library[91]. `gmm` uses an *expression* template implementation involving the close linking of a hierarchy of linear-algebra *traits* structures with classes implementing the underlying storage containers themselves. Expression templates are instantiated at compile time rather than run time and thus entail minimal run-time overhead. In fact, expression templates can avoid overhead that would otherwise be associated with the function calls themselves. In practice it turns out that, at double precision, the `gmm` BLAS and higher-level linear algebra methods are faster than the standard BLAS and LAPACK methods used by most non-templated numerical-computation libraries.

The linear-algebra traits mechanism used by `gmm` allows the library to be easily extended while retaining use of already-defined classes and functions. Several such extensions required for the code-base include:

- Sparse vector and matrix classes, `indefiniteVector` and `indefiniteSparseMatrix`, designed for representation of very high-order sparse matrices like those involved in T-matrix calculations for very large structures requiring coefficients at all multipole orders but only a few values of azimuthal degree.
- Block-vector and block-matrix classes, `block_resident_vector` and

`block_resident_matrix`, able to represent structures too large to fit into high-speed memory, thus allowing computation with extremely large structures. (This extension was originally intended just for use with dense matrices, but it also works correctly with sparse-matrix blocks.)

- A fixed-length vector class, `ntuple`, designed to efficiently represent coordinate points.
- N-dimensional matrix classes with associated interpolation methods.
- Singular-value-decomposition (SVD), QR-factorization, and null-space algorithms.

An important planned code-base extension needed for interfacing to linear solvers beyond those discussed in Section 5.3.5 will require enhancing the `gmm`-interfaced block-matrix processing capabilities so that they can simultaneously utilize multiple processes, spanning multiple compute nodes in a computational cluster. Presently a linear-solver multi-process utilization functionality is realized through use of the `TRILINOS` library[92], but dependence on this library must eventually be phased out because of its incomplete templating and because of its many `FORTRAN`-library dependencies.

5.2.4 Parallel processing support

5.2.4.1 Thread-level parallelism

Currently there are two main approaches used to achieve thread-level parallelism in C++ programming, first, use of a low-level threads application programming interface (API) such as the `POSIX` threads API `PTHREADS`[104] and, second, use of the `openMP` compiler pragmas[105]. Other approaches do exist, for example

Cilk++, which introduces new language keywords[106, 107]. However, it is not clear to what degree these other approaches will be adopted by the open-source community, and it is this community that is largely responsible for advances in the C++ language and its associated tool-chain.

At the time when development work commenced on the code-base herein discussed, the openMP implementation on the GNU compiler tool-chain[86] was incomplete while the one on the alternative INTEL compiler tool-chain[87] was inconsistent with the openMP standards specification. After an extensive testing sequence designed to elucidate the specifics of interactions between openMP and the C++ language, with extremely inconsistent results, it was decided not to use either of these implementations, and it was further decided to simply *define* an interface that would enable consistent behavior and that would support thread-level parallelism features equivalent to those provided by the openMP API. This interface is implemented using the template functions OMP_parallel_for in name-space parallelUtil (see section 1.6.9). The functions are implemented using the PTHREADS API and are used to provide thread-level parallelism throughout the code-base. However, this solution is not ideal, both because of the increased complexity that the interface adds to the code-base and because of the likelihood that a native compiler-based solution will use computational resources more efficiently. As time allows, the interface will gradually be re-implemented to make use of native openMP pragmas, the implementation of which has hopefully improved in the interim.

The primary difficulties associated with multi-thread parallel programming relate to data structures shared among threads. There is in general no problem associated with *immutable* data structures, but for *mutable* data structures the possibility exists that one thread will modify a structure in a way that invalidates another thread's current activity that is using the previous value of the structure, or several threads

may attempt to modify a structure at the same time. When operations on a data structure by a thread can be *atomic* with respect to the possibility of interruption by another thread such problems do not occur[108, 109]. The most common technique for ensuring atomic operations involves the use of a *mutual exclusion lock*, or *mutex*, which is a special type of state variable that is provided by the low-level thread API and that can be *locked* by only one thread at a time. Prior to using any mutable data structure a thread must lock the mutex associated with the structure, and when the usage is finished the thread must unlock the mutex. The overhead cost of this approach depends on the number of mutexes used and on the frequency of attempts by threads to lock a mutex already locked by another thread. The operation of locking a mutex that is presently unlocked accrues minimal overhead[104].

For large data structures such as matrices the probability, that threads will attempt to operate on the same structure component at the same time, decreases inversely with the overall size of the structure, i.e., in the case of a matrix, with its order. For computations involving such large structures, associating a single mutex with the entire structure unnecessarily prevents threads from working on different parts of the structure at the same time. A solution to this problem involves associating a set of mutexes with a corresponding set of keys, where the size of the two sets corresponds to the maximum number of threads to be allowed to operate simultaneously on the structure. Prior to undertaking an operation on a structure, a thread must lock a mutex and set, as value of the key associated with the mutex, an identifier of the particular structure part to be operated on. Different threads will usually be working on different parts of the structure so usually the lock can take effect and the thread can immediately proceed with the operation, with the locking activity thus entailing minimal overhead. In rare cases, some other thread will already be operating on exactly the same part of the structure, as indicated by the key value, and the thread will

need to wait before it can lock the mutex. The additional overhead required for use of this multiple-mutex scheme is determined by the maximum number of threads allowed to be active for a particular task, a number that is often the number of available CPU cores and thus that is often quite small for currently available computing hardware. This multiple-mutex scheme is provided by the class multiMutex (see Section 1.7.133) which implements mutex-keying operations suitable for use with most data types.

5.2.4.2 Process-level parallelism

The most widely-used implementation technique for enabling computer programs to activate inter-communicating multiple processes running simultaneously on different CPU cores, or possibly even on different computers, uses a standardized message passing interface called `MPI`[110]. This is the technique used in the codebase herein discussed. MPI consists of a set of primitive functions and associated data structures providing a wide range of inter-process synchronization and communication capabilities. Although there are many different vendor-specific and open-source implementations of MPI, by and large, programs using this interface are portable across different platforms.

The process-level parallelism currently enabled for the multiple-particle T-matrix computation envisions two distinct usage scenarios:

- Performance of a number of tasks where during task execution itself no inter-process communication occurs among processes performing any of the particular tasks. This is the scenario of the present multi-centered implementation (see Section 5.3.4) where T-matrix computation at each wavelength value is assigned to a distinct process.

- Performance of a number of tasks where high-bandwidth inter-process communication occurs among processes performing the particular tasks. This is the scenario of the present cluster-centered implementation (see Section 5.3.5) that applies a process-distributed linear-algebra solver to optimize the implementation for aggregate structures containing a very large number of particles.

Having only a small set of scenarios allows the use of a *dispatcher-receiver* scheme where a single *dispatcher* process assigns tasks to multiple *receiver* processes, monitors the progress of the resulting computations, and assembles results as they are completed by the receivers. Allowing for the possibility that there be only one receiver allows a single such scheme to handle both of the scenarios just described. This scheme is realized by the dispatcher and monolithic_dispatcher classes (see Sections 1.7.123, and 1.7.130, respectively) in name-space parallelUtil (see Section 1.6.9). Implementation of a particular calculation requires specialization of the `task` and `result` components of the classes followed by definition of an `executeTask` method.

With respect to inter-process communication, it should be observed that a commonality exists among the following kinds of data-transfer operations:

- Transfer from one type of memory to another type, such as from resident to out-of-core memory.
- Transfer from one memory location to another.
- Transfer from memory controlled by one process to memory controlled by another, possibly even over an external communication channel.

In modern programming practice, the first of these operation kinds is almost always defined for any high-level data-structure type because it is required for transferring an instance of the type to a *persistent* memory (e.g., to a file on a hard disk) for later re-use. Implementation for a data-structure type of the second kind of transfer makes it possible for generic sub-routines to transfer an instance of the type even though they have no access to its internal structural details. This facilitates *modularization* of code thereby enabling easier code maintainability. Implementation for a data-structure type of the third kind of transfer enables an abstract, high-level treatment of process-level parallelism. Correct treatment of such process-to-process transfer is one of the most difficult aspects of process-level parallel programming.

When a user needs to specify a class determining a new data-structure type, the abstractCommHandle class and the associated primitive functions in the commUtil name-space (see Sections 1.7.2 and 1.6.1, respectively) enable a unified treatment for the type of these different kinds of transfer operations. All that is necessary is for the class to implement three methods: `readBinary`, `writeBinary`, and `binarySize`. The first two implement binary I/O operations and the third determines how much resident memory is occupied by a particular instance of the class. The primitive functions `open`, `close`, `read`, and `write` in the `commUtil` name-space all have calling signatures corresponding directly to the system functions with the same names, and so the use of these primitives in the construction of `readBinary` and `writeBinary` is straightforward (although an `abstractCommHandle` is used instead of a pointer to a file structure). Once this implementation is completed for a given class, instances of the class can be efficiently transferred using exactly the same functions, first, to and from a file, second, to and from a memory buffer, and, third, to and from a process running on a separate processor, even one that may be part of a separate computer.

It is notable that, by using the `dispatcher` class for implementation of inter-process synchronization and by using the `abstractCommHandle` class for implementation of inter-process communication, essentially no `MPI` function calls need appear in codes realizing T-matrix calculation specifics. Thus the difficult problems usually associated with achieving process-level parallelism are in general avoided in the development of new T-matrix calculation codes.

5.3 Problem-space specifics

5.3.1 Indexing on the vector multipole basis

In order to make use of the basic linear algebra subprograms (BLAS) interface provided by `gmm`, the dual (l, m)-index scheme for the VSW must be mapped to a single linear index. This is accomplished with a set of utility functions provided by the tmatrixIndex class (see Section 1.7.169), within the TMatrix name-space. The following index mappings are available:

$$x \leftarrow l^2 + l + m \qquad \text{contiguous-}m \text{ multipole index} \tag{5.7}$$

$$x \leftarrow l \qquad \text{monolithic-}l \text{ index} \tag{5.8}$$

$$x \leftarrow \begin{cases} l - m^2 - (l_{\max} + 2)m - 1 & : m < 0 \\ l - m^2 + l_m ax(m + 1) + m & : m \leq 0 \end{cases} \quad \text{contiguous-}l \text{ multipole index} \tag{5.9}$$

where x is the single linear index, and l and m are the multipole order and azimuthal degree, respectively. For each mapping above, also provided is the inverse mapping which takes $(l, m) \leftarrow x$ given a specified value for the maximum multipole order $l_{\max}$. Yet more mappings add hierarchy corresponding to the types of transverse VSW, either $\vec{M}$ or $\vec{N}$, and to the field regularity, which may be any of `REGULAR`, `OUTGOING`, or `INCOMING`.

5.3.2 Translation operator for the VSW basis

The initial implementation of the VSW propagator was based on the formulas from Borghese et al. [34], although the propagator was factored to allow its representation in terms of separate rotation and single-axis translation operators. For this particular implementation the required rotation operator was constructed using the Jacobi functions (see Section 1.7.91). Although the Borghese formulas are precisely correct, their complexity exacerbated implementation difficulties. In particular, the Gaunt coefficients must be evaluated and this required development of the gauntFunctor class (see Section 1.7.88) along with associated support classes that provide multipole-sextuple indexing and iteration (see Section 1.7.96). In fact, in this case the factoring of the propagator was required mainly to reduce the number of Gaunt coefficients needed by the calculation, which number would otherwise have been prohibitively large.

As part of the verification of these propagator codes, the recurrence-relation based propagator design proposed by Mackowski[78] was additionally implemented, and this implementation turned out to be so much simpler and more efficient that the original propagator implementation is no longer used in production code. However, the original propagator implementation does provide important validation capability for any VSW propagator or rotator and thus is retained in the propagatorTMatrix (see Sections 1.7.165) and rotatorTMatrix (see Section 1.7.166) classes.

As mentioned in Section 4.3, factoring the VSW propagator into separate rotation and translation operations significantly decreases computational complexity. This factorization is encapsulated in the factoredPropagator class (see Section 1.7.153) which is used by both the multi-centered and the cluster-centered multiple-particle T-matrix calculations. For use internal to the `factoredPropagator` class, the recurrence-relation based implementation of the rotation operator is provided

by the rotationCoefficientFunctor class (see Section 1.7.108) and the recurrence-relation based implementation of the single-axis translation operators is provided by the additionCoefficientFunctor class (see Section 1.7.68). For completeness the `additionCoefficientFunctor` can optionally calculate either the scalar or the vector translation coefficients.

The matrix representations of the rotator and of the single-axis translator are most efficiently stored using the contiguous-l and the monolithic-l indexing schemes discussed in section 5.3.1. To optimize memory use as well as use of other run-time resources, special block-sparse *skyline* vectors and matrices are used for the storage of these propagator components[91], and special versions of the vector and matrix BLAS functions have been implemented to be used internally by the `factoredPropagator` class. These functions provide both left and right multiplication by these component matrices, taking advantage of the fact that the rotator and the single-axis translator are diagonal in the multipole order l and in the azimuthal degree m, respectively.

5.3.3 Representations of the T-matrix: single-particle, multi-centered, and cluster-centered

As discussed in Section 5.2.2.2, the initial applications of the code-base required a very-high-order sparse matrix representation of the single-particle T-matrix. This representation is implemented in the `gmm`-interfaced indefiniteVector and indefiniteSparseMatrix classes in the linalg::sparse name-space (see Sections 1.7.47, 1.7.46, and 1.6.5, respectively). This sparse representation would be quite inefficient if used during the multiple-particle T-matrix calculation itself and so for this purpose *dense* and *skyline*[91] matrix-storage representations are used. These dense multi-centered T-matrix representations are block-matrix structures where

the blocks are indexed by field-class (i.e., VSW $\vec{M}$ or $\vec{N}$) and field-regularity type (i.e., `REGULAR`, `OUTGOING`, or `INCOMING`). This block-matrix implementation is provided by the blockVector and blockMatrix classes in the linalg::sparse name-space (see Sections 1.7.43, 1.7.43, and 1.6.5, respectively). These latter classes are in the `linalg::sparse` name-space because the occupancy rate of the blocks themselves is sparse. Actually the sub-block structure itself can be any of the many types of `gmm`-interfaced matrix classes, and can use either a sparse or dense storage scheme. With respect to the cluster-centered T-matrix representation specifically, after the calculation is complete the T-matrix is indistinguishable in terms of representation from any single-particle T-matrix. In fact, the only distinction from a single-particle T-matrix at this point is that the cluster-centered *scattered-fields* T-matrix has no corresponding T-matrix representing the *internal-fields* of the structure.

5.3.4 Recursive, multi-centered implementation

As indicated previously, given the single-particle T-matrices for the aggregate's component particles and a suitable representation for the VSW propagator, the multiple-particle T-matrix calculation is essentially a linear algebra problem. (For the multi-centered calculation, the mathematical details have been discussed in section 4.4.3.) This linear-algebra calculation proceeds by recursively constructing the block matrix that will be the solution, incrementing the aggregate structure by one particle during each recursion step. The increment for each additional particle requires both modification of the previous solution blocks and addition to the solution matrix of a new block row and column. At completion, this algorithm will have performed $\mathcal{O}\{N^3\}$ matrix multiplications and N matrix inversions, where N is the total number of particles in the structure.

The specific recursive technique used to evaluate the cluster-centered T-matrix can be very efficient and it potentially treats, better than other approaches[83], issues relating to the poor condition numbers for the matrices representing the *irregular* VSW propagator. However, validation studies comparing the condition numbers for intermediate matrices participating in this calculation, with the condition numbers for analogous matrices participating in the cluster-centered calculation, found that the two approaches are actually quite comparable in this regard. In addition, the fact that the size of the block-matrix system gradually increases during the cluster-centered calculation makes it quite difficult to efficiently scale this approach to multiple processes. The resulting lack of an efficient distributed-processing implementation for this algorithm limits the maximum number of particles that can be treated with this approach to approximately 500 (depending on the maximum single-particle multipole order used in the calculation). It is possible that use of an alternative linear-solver algorithm would result in less constraining limitations and so plans are underway to perform comparison studies. Use of modular techniques to encapsulate interface details relating to the specific solver alternatives, which techniques have already been developed as part of the cluster-centered calculation implementation, will allow these studies to proceed rapidly.

In its current implementation, unlike the cluster-centered solution, the multi-centered solution provides no estimate for the accuracy of the calculated results. Of course such an estimate can be arrived at by comparing results from multiple runs with gradually increasing single-particle multipole order $l_{\max}$, but for large structures the computational resources required for these multiple runs are prohibitively expensive. In principle it should be possible to implement a technique for evaluating the accuracy of the solution with respect to the extinction efficiency Q_e in analogy to the approach used by the cluster-centered algorithm. However, this is

less straightforward for the *recursive* multi-centered solution because the intermediate values of Q_e for this solution are actually from *different* structures. This gives additional justification for changing the multi-centered implementation so that it uses a more traditional linear-solver approach.

The multi-centered variant of the multiple-particle T-matrix calculation is provided by the multiCentered_tmatrix class (see Section 1.7.159) which also implements the most commonly used cross-section methods, including methods for evaluating the partial cross sections. (This particular T-matrix calculation scheme is the only one that makes these partial cross sections available.)

5.3.5 Cluster-centered implementation

The mathematical details of the cluster-centered calculation were discussed in Section 4.4.2. This linear algebra calculation is initialized by constructing the block-matrix left-hand side (LHS) and the block-vector right-hand side (RHS) corresponding to the structure elaborated in Section 4.4. Using the *Debye truncation criterion* (see Section 2.2.2.1), maximum multipole orders $l_{1:\max}$ and $l_{2:\max}$ that correspond to the circumscribing spheres for the individual particles and for the entire cluster, respectively, are estimated. The single-particle T-matrices used for initialization are sized symmetrically using the single-particle $l_{1:\max}$ as the truncation criterion for both the rows and the columns. However, the VSW propagators used in the initialization (and during the calculation itself) are asymmetric with respect to maximum multipole order, with the single-particle $l_{1:\max}$ being used for the column truncation criterion and the cluster $l_{2:\max}$ being used for the row truncation criterion[3]. This calculation could ultimately produce a cluster-centered T-matrix

[3]In this discussion, it is to be understood that *row* and *column* are to be reversed when the propagator is used for *right* application to a T-matrix.

with a symmetric row and column truncation criterion corresponding the the cluster $l_{2:\max}$ but, in practice (as indicated in Section 4.4.2), a column-by-column construction of the solution allows an ongoing determination of the degree to which the extinction efficiency Q_e has converged to its final value. Provided that the truncation criteria have been estimated correctly, in many cases columns far fewer in number than the number corresponding to the cluster $l_{2:\max}$ are actually required to obtain convergence.

Further discussion is required with respect to the selection of a suitable linear algebra *solver* implementation. The primary classification of linear solvers distinguishes between, on the one hand, *direct* solvers that calculate a solution to the maximum accuracy possible given the precision of the number system used and thereby produce the equivalent of a matrix factorization of the LHS and, on the other hand, *iterative* solvers that for a particular RHS column (or a block of columns) gradually arrive at a solution through an iterative convergence. Even though more than one solver cycle may be required to obtain results involving multiple RHS columns, the required accuracy of the solution is in many cases much less than the maximum possible, and so the computation resource requirements for application of an iterative solver to a given problem can in turn be much less than the requirements for application of a direct solver to the same problem.

Because a large number of both direct and iterative solver implementations are available in the public domain, it was desirable to use a *modular* linear solver implementation approach, thus allowing one solver to be easily substituted for another when required for comparison, verification, or other purposes. Such an approach is realized with the class distributedSolverFunctor that internally uses the class distributedSolverModule (see Sections 1.7.78 and 1.7.8, respectively). All that is required to use a particular linear solver is to construct an appropriate spe-

cialization of `distributedSolverModule`. At present the particular linear solver used by the cluster-centered T-matrix calculation is the block-GMRES solver available within the `Trilinos::Belos` name-space, which is part of the `Trilinos` library released into the public domain by Sandia National Laboratories [92]. Important benefits of solvers available within the `Trilinos::Belos` name-space are that they can make use of several different *block* solver schemes for processing multiple RHS columns simultaneously and that many of the solvers are fully scalable across multiple-processor implementations using MPI. Drawbacks of the `Trilinos::Belos` implementation result from the fact that it is not fully templated, it can work only with real numbers, and it has implicit dependencies on FORTRAN source code and libraries.

The cluster-centered variant of the multiple-particle T-matrix calculation is provided by the clusterCentered_tmatrix class (see Section 1.7.143). Cross-section methods associated with this class are provided by the class tmatrix (see Section 1.7.168) which is the class that implements the single-particle T-matrix calculation and associated methods.

5.3.6 Full-fields calculation

Given a vector of VSW coefficients representing the electrodynamic fields with respect to a single origin, and given an observation point, the calculation of the full electrodynamic fields requires the accumulation of the contributing VSW components at each participating multipole index (l, m). For a vector of multi-centered VSW coefficients, this accumulation must be continued until it includes contributions from each participating VSW basis. The only important implementation details relate to optimizing the accumulation sequence in order to minimize the number of special-function evaluations required.

For the naive calculation, each point on the mesh requires $\mathcal{O}\{N_p l_{\max}^2\}$ special-function evaluations, where N_p is the number of particles in the aggregate. However, using in combination with a simple nearest-neighbor interpolation scheme, the fact that each VSW component is composed of terms *separable* in the spherical coordinates, the number of required evaluations can be greatly reduced. In effect this approximates the field contribution at each mesh point using the field at a point nearby on a separable spherical coordinate grid. Provided that the spherical coordinate grid is sufficiently fine, this approximation can be as accurate as desired. This "jitter mesh" approach provides almost a 100-times speed-up in typical field calculations involving field evaluation at grid points on a 2D plane. Specifically, considering only special-function evaluations, a calculation involving N^2 grid points can be effectively reduced to a calculation involving only $\mathcal{O}\{N\}$ points. Unfortunately, additional unavoidable overhead scales with the number of grid points and reduces the overall benefits of this approach but, regardless, its benefits can be substantial.

Full-fields calculations are provided by a special numerical functor, the fieldVisualizer class (see Section 1.7.86) which, to evaluate the contributing VSW components at each observation point, uses the `apply` methods of the vectorFieldKet class (see 1.7.170). These methods in turn use the vectorSphericalHarmonicFunctor class for this task (see Sections 5.2.2.6 and 1.7.119).

5.4 Future directions

Current plans are to release the code-base into the public domain within a year. The release will be accompanied by a publication that provides an overview of the code-base, a verification and demonstration of key capabilities, and a comparison with other existing codes serving similar areas of application. The most difficult aspects

of this planned release relate, first, to the modifications necessary to make the code-base more accessible to less computationally experienced users and, second, to the reduction of the complete code-base to a size easier to support and maintain but that nevertheless continues to possess the capabilities and features most frequently needed and/or desired by the code-base's intended users.

Use of multiple representations for the T-matrix itself (see Section 5.3.3), with the one used for a particular calculation depending on the calculation type and possibly even changing from one stage of the calculation to the next, makes the code-base much more complicated than need be. For the pending release of the code-base into the public domain, mainly because of maintenance issues associated with that release, it will be desirable to have a uniform representation for the T-matrix. Most probably the dense matrix representation will be selected because, first, most single-particle T-matrices are likely to have too small a memory footprint to accrue significant benefits from a sparse representation and, second, multiple-particle T-matrices are almost always comprised of dense coefficients, except in a few very special cases.

The numerical functor class has been in use for several years as part of production code. During this period there have been many times when difficulties of code development and maintenance have been exacerbated by, rather than assisted by, the numerical functor design for status and error reporting. Major problems are caused by the fact that the current design does not specify mechanisms for *automatic* enforcement of error-reporting precedence. In a better design, status messages associated with the first point where any error is detected would always be maintained and any additionally available calling-sequence related error messages would be *automatically* appended to these initial messages. A second problem has also become evident, and that is that there is in the current design no way to require a user to even

check a functor's return status when an error occurs, let alone record that return status, and this will potentially become a serious problem when the code-base is used by less experienced programmers. For these reasons the numerical functor error and status reporting design will be modified so that it specifies a mechanism for enforcing these requirements. An implementation using the C++ exception mechanisms rather than the functor return value will enable a straightforward solution to this problem.

Although the C++ exception mechanisms are often misunderstood and thereby maligned by programmers more familiar with the other programming languages traditionally used for numerical computation, modern implementations of these mechanisms do not unduly increase run-time overhead or affect code execution speed. It is actually possible for C++ compilers to implement these mechanisms without run-time speed overhead[111], although in practice there is usually an overhead near 10%[112], which it should be possible to reduce through use of advanced coding techniques that pay special attention to stack utilization during function calls. In any case, a slight decrease in run-time speed is a small price to pay for a large increase in code maintainability thus allowing programmers, whose time is generally a most critical resource, to be more productive.

During the evolution of the code-base, the C++ language itself has been evolving. Of particular relevance to the numerical functor implementation are recently proposed extensions to the C++ standard library, as specified in *C++ Technical Report 1* (TR1)[113]. These extensions include implementations of many of the mathematical special functions now implemented as parts of the numerical functor classes of the code-base herein described. Although support for TR1 extensions across different compilation platforms is highly variable at present, many TR1 extensions are available as part of the `Boost` C++ libraries[114], and thus it should be possi-

ble to make use of at least some TR1 extensions immediately. Because the code-base's numerical functor classes enable many capabilities beyond the evaluation of the mathematical special functions, much of the current `numericalFunctor` implementation will remain in place, but gradually the `apply` methods used in `numericalFunctor` instances will be modified to make use of TR1-based implementations. Because these modifications will reduce the size of several critical components of the code-base, they will improve code-base maintainability.

Chapter 6

Advanced phenomenology

6.1 Introduction

The simple phenomenological rules presented in Chapter 3 applied a comparative approach to predicting expected spectral differences among the optical cross sections of aggregates with differing structures. The focus was largely on predicting the spectral shift of a single peak in the cross section, and this approach has been successful for explaining observations from live-cell labeling experiments[115]. However, many experimental results show more complicated peak structures and, in fact, such structures are even evident in the series of numerical simulation results presented in Chapter 3.

The present chapter applies the analytic framework of the multiple-particle T-matrix technique to a more comprehensive exploration of the electrodynamic behavior of aggregate structures. As with the preceding discussion, the focus will be on the development of qualitative rules for describing the spectral properties of the optical cross sections. This approach includes direct evaluation of the spectral properties associated with *collective* modes relating to the aggregate structure as a whole as well as with *per-particle* modes, and in addition includes exploration of the relationship between near-field inter-particle interactions and far-field observations.

To facilitate the analysis, numerical simulation results will be presented for a selection of representative aggregate systems. This will demonstrate the validity of more

general conclusions for these systems specifically but it is intended also to increase plausibility of the assertion that the conclusions apply to more general systems. Although it is believed that the concepts presented here are generally applicable, further proof is beyond the purview of the present discussion.

6.2 Collective mode interactions

During discussion of the properties of the single-particle T-matrix in Chapter 2, it was observed that Debye's analysis of the dependence of the Mie coefficients on particle radius[4] also applies directly to the T-matrix coefficients for particles defined by arbitrarily-shaped boundary surfaces[24] (see Section 2.2.4). This extension of Debye's analysis is largely based on consideration of the radial dependence of the *irregular* portion of the VSW basis functions at the boundary surface. Such a function has exponential behavior when the radius, expressed in optical units, is sufficiently less than the multipole-order of the function. The implication is that the single-particle T-matrix coefficients corresponding to these higher multipole-orders, as well as the magnitude of the associated far-field components of the scattered fields, are exponentially suppressed.

Using the cluster-centered form of the multiple-particle T-matrix (see Section 4.4.2), it is possible to evaluate directly the contribution of a specific *collective* multipole-order to the scattered fields from a multiple-particle structure. However, despite the formal equivalence between this T-matrix and any T-matrix representing the electrodynamics of a single particle, due to the fact that the cluster-centered T-matrix does not represent a system with uniform internal structure, care needs to be taken in applying the single-particle results regarding the behavior of the T-matrix coefficients. To extend the single-particle results to the cluster-centered T-matrix, it is sufficient to consider the outgoing-wave expansion of the scattered

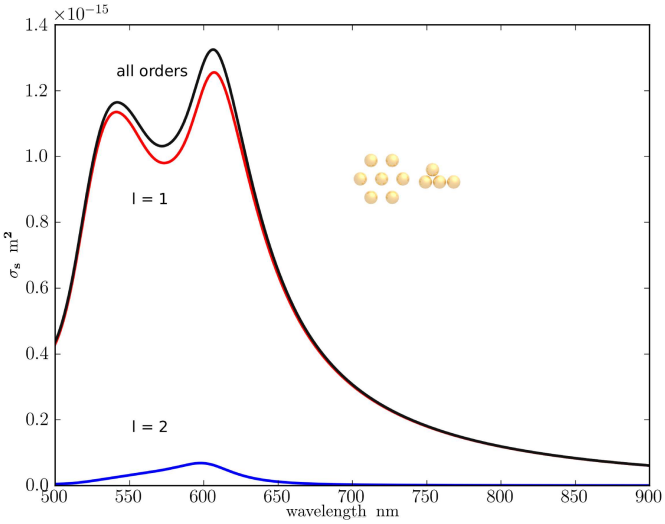
fields expressed at a spherical surface of radius R circumscribing the structure. Then (assuming angular features in the expansion are roughly of the same magnitude with respect to multipole order), it is seen that a necessary implication of the exponential dependence of the basis functions is that high multipole-order (where $kR \ll l$) angular details will be strongly suppressed at radial distances greater than the multipole-order l. This then is an alternative way of looking at Debye's "multipole truncation criterion," a way that also applies to the coefficients of the cluster-centered T-matrix.

At the resonance wavelength for a small gold nanoparticle, about 535 nm , one *optical unit* corresponds to about 85 nm . This means that, except for the very largest structures, the aggregates considered here typically have a size on the order of one optical unit and thus predominantly the collective dipole components account for the largest part of the scattered fields. Using the cluster-centered T-matrix, the dipole and quadrupole contributions to the scattering cross section for the example structures considered in the following sections are shown in Figures 6.1 and 6.2, and the relative magnitudes of these contributions are fully consistent with this analysis. The differences between the relative magnitudes of the quadrupole component for the two structures is due to the asymmetry of the structure of Figure 6.1, which asymmetry enhances the quadrupole contribution. Finally, and most importantly, it will be noted that the detailed sub-features in the scattering cross-section spectra are already appearing in the dipole-only cross sections and thus cannot be attributed to distinct collective modes.

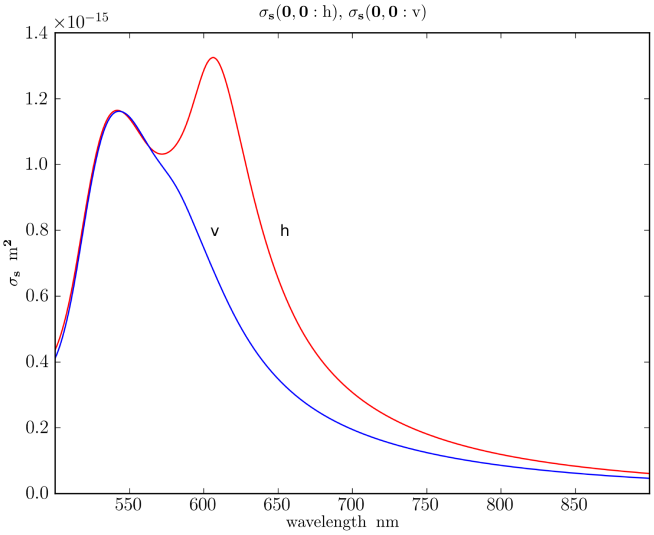
For completeness, the polarized cross sections are also shown in the second panel of each figure and, from the perspective of collective-mode analysis, these correspond to details of the $m = \pm 1$ *azimuthal* components of the scattered multipole fields. The spectral differences between the two polarized cross sections for the

more asymmetric structure are well explained by the simple considerations of the angular symmetry of the dipole fields and of the effect of this symmetry with respect to changes in structure aspect ratio (discussed in Section 3.2.3.2).

Finally, although for the specific aggregate structures considered here it has been shown that collective-mode analysis cannot account for the details observed in the scattering spectra, this conclusion would not be correct for many fabricated aggregate structures considered in the literature[116, 117, 118, 119]. For example, for structures fabricated using electron-beam lithography, typical single-particle dimensions are presently of about 100 nm and thus structures containing even a few particles already span *multiple* optical units. For such structures it will often be the case that features of cross-section spectra can be attributed to particular collective multipole modes.

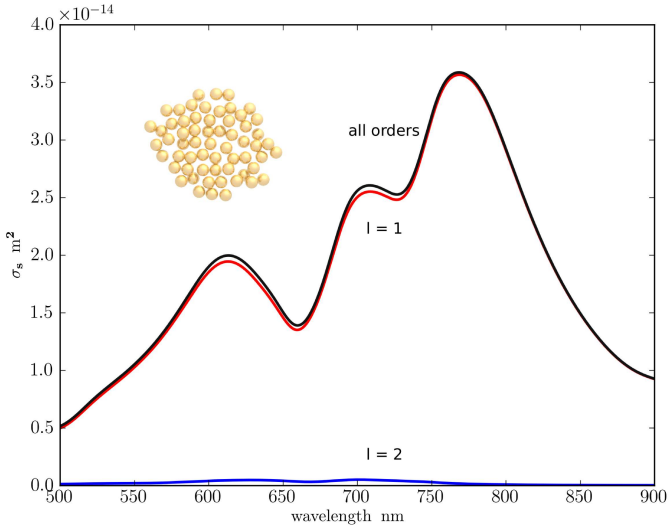


(a) Dipole and quadrupole contributions to the unpolarized scattering cross section.

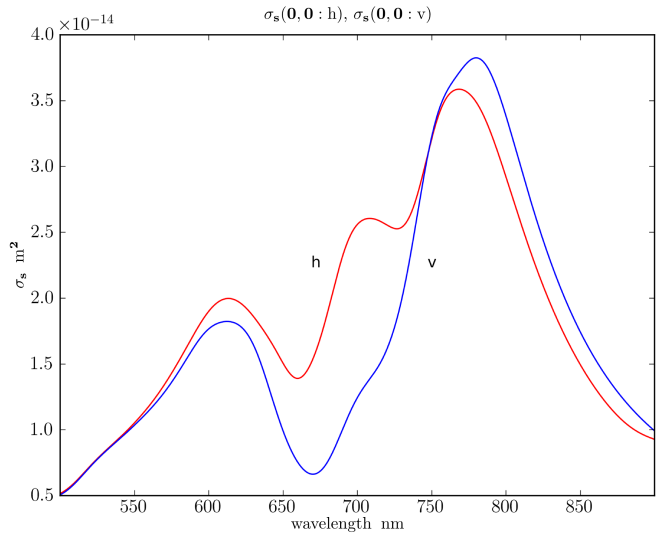


(b) Polarized scattering cross sections.

Figure 6.1: Collective mode contributions to cross sections for the structure to be analyzed in Section 6.3.1.1 (25 nm particles, a = 2.1).



(a) Dipole and quadrupole contributions to the unpolarized scattering cross section.



(b) Polarized scattering cross sections.

Figure 6.2: Collective mode contributions to cross sections for the structure to be analyzed in Section 6.3.1.2 (25 nm particles, a = 2.1).

6.3 Per-particle interactions

Using the multi-centered form of the multiple-particle T-matrix (see Section 4.4.3), it is possible to evaluate directly the per-particle contributions to the electrodynamics of the structure. In principle the most straightforward manner to evaluate directly the per-particle electrodynamics would be to calculate the complete electrodynamic fields throughout the structure and then to interpret these results. Although this approach will be applied to a specific problem in Section 6.4, in many practical cases the approach provides so much information that the interpretation of these full-field results can be quite difficult. An alternative is to use information available through various reductions and superpositions of the dual-centered T-matrices forming the complete multi-centered T-matrix. The most commonly used examples of such reductions are the expressions for the optical cross sections themselves, but these particular reductions provide only information about collective behavior. An alternative reduction that is specifically relevant to the evaluation of per-particle effects on cross-section spectral properties is the *partial* cross section (see Section 4.4.3.2). The particular form of partial cross section applied here represents the power contribution *transferred* by a single particle within the structure to the total scattered power for the entire structure, normalized by the intensity of the incident fields. Although the total power scattered by the structure must be non-negative, this is not true with respect to power transfer within the structure. That is, individual partial cross sections can be and often are *negative*. Evaluation of the spectral properties of these partial cross sections then allows direct evaluation of the per-particle contributions to the properties of the complete cross section for the entire structure. To illustrate different types of per-particle effects, several examples of this technique will be applied in the rest of this section.

6.3.1 Constructive superposition

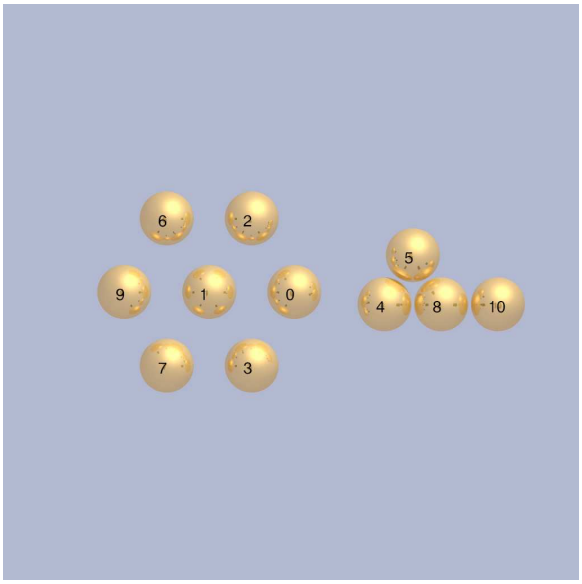
Using considerations analogous to those of Section 3.2.1, it may be possible to partition an aggregate into sub-groups of particles such that the induced near-field coupling between sub-groups is minimal. In this case, apart from the coherent superposition of the scattered fields, these sub-groups will behave independently, and cross-section spectral features can be directly associated with the superposition of the spectral features from each sub-group. Given a known structure, in some cases the attribution of spectral features will be intuitively evident through considerations of the morphology of particular sub-groups. For example, if a particular sub-group has a smaller mean inter-particle spacing compared to another, it can be associated with a longer-wavelength spectral feature in the complete cross section. However it turns out that in many cases, even with a known structure this attribution is not at all straightforward. Therefore, to develop insight applicable to these cases and also to cases where the specifics of the structure are unknown and only spectra are available, a more systematic approach for attribution will be constructed.

6.3.1.1 A simple structure

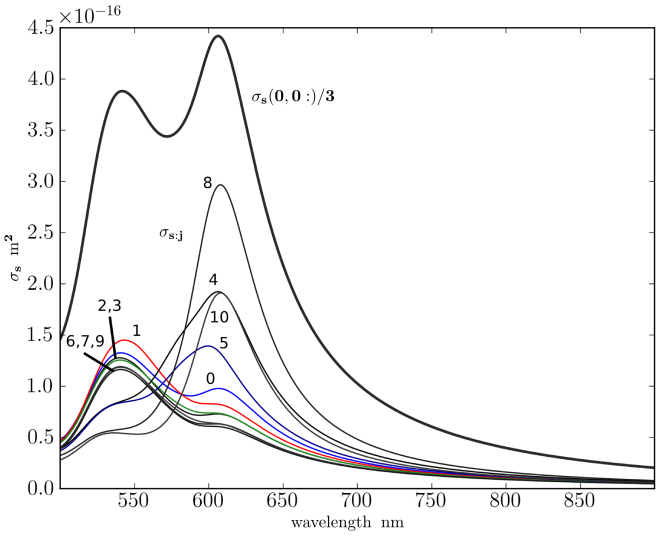
To illustrate the important concepts, a particular example structure will be analyzed, a structure that has been designed to include multiple, regionally-localized sub-groups of particles but that has a large enough spacing between the sub-groups to minimize the effects of inter-group coupling. The scattering cross section for the structure shown in Figure 6.3a does in fact manifest two distinct peak features. Further, the corresponding partial scattering cross-section plots of Figure 6.3b allow the direct per-particle attribution to these features, and this validates the attribution that might have been made from considerations of the mean inter-particle spacing within each sub-group, that is, that the heptamer is associated with the shorter-

wavelength feature and the tetramer with the longer-wavelength feature.

To facilitate this type of analysis for structures containing a larger number of particles, each partial cross-section spectrum is converted to a pseudo-color by multiplication of the spectrum with specified Gaussian distributions for each of the red, green, and blue color channels and then performing an r.m.s. integration. Details of this pseudo-color generation step are illustrated in Figure 6.4a which shows the specific color channel distributions used to analyze the present structure. In this particular case and in all color-mapped figures shown in this chapter, the specifics of the color channel distributions used are selected "by hand" in order to maximize the color variation and thereby the displayed information. These pseudo-colors are then used to label a representation of the aggregate structure as shown in Figure 6.4b. In this illustration the attribution of the shorter-wavelength and longer-wavelength peak features as arrived at from examination of the individual partial cross-section plots is now explicitly clear and, in addition, other interesting details illustrate subtle differences in the degree of inter-particle coupling throughout the structure.

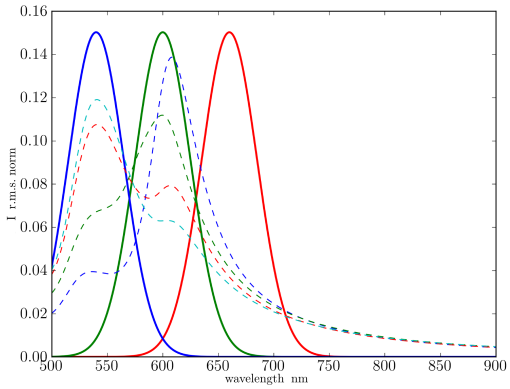


(a) Representation of aggregate structure.

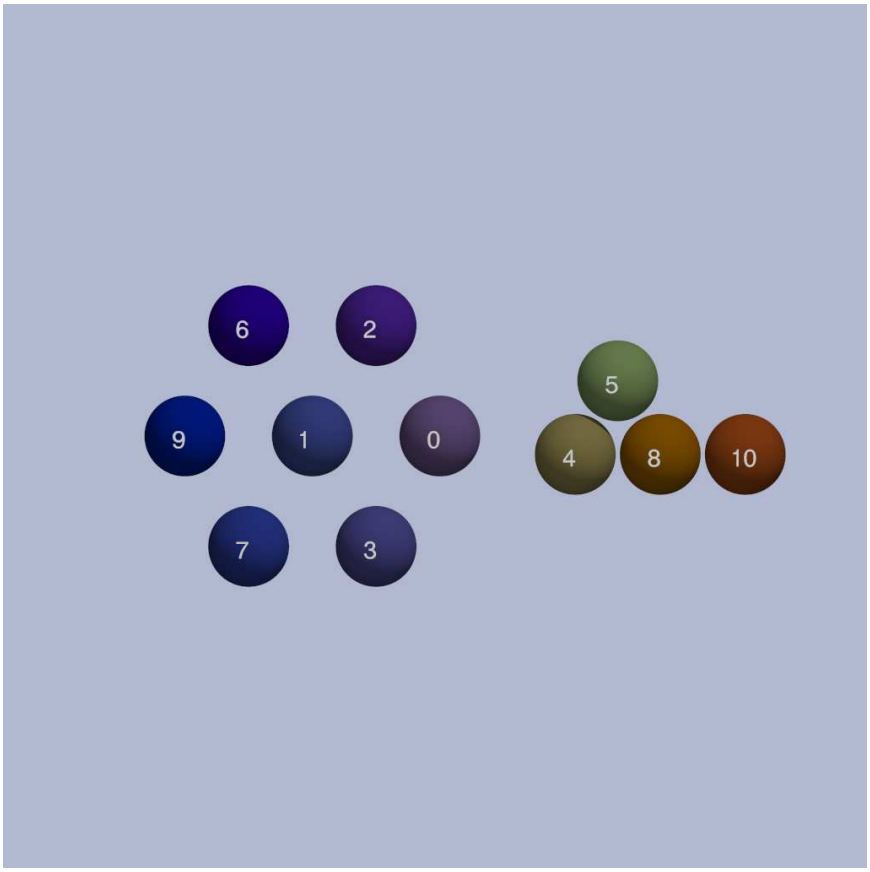


(b) *Complete* and *partial* cross sections; particle numbering as indicated (25 nm particles, a = 2.1).

Figure 6.3: Example aggregate system including particle sub-groups.



(a) Color channel distributions



(b) Pseudo-color labeling of aggregate representation.

Figure 6.4: Aggregate representation labeled using colors generated from partial cross-section spectra.

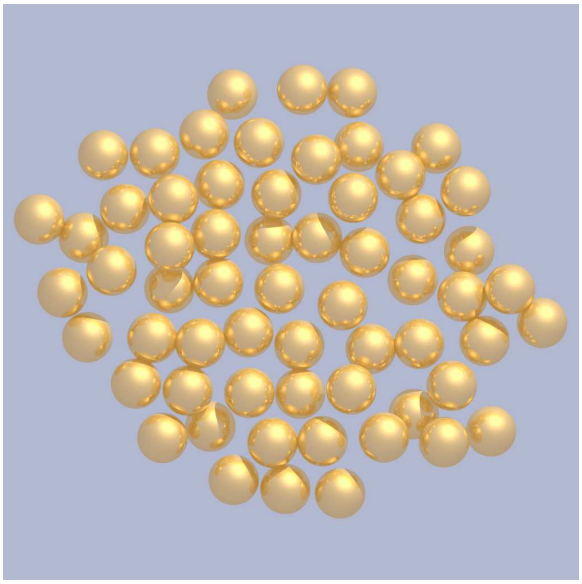
6.3.1.2 A more realistic structure

Partial cross-section analysis will now be applied to a structure that has a morphology quite similar to that observed in aggregates produced during live-cell labeling experiments (see Section 3.1). Examining the representation of the structure shown in Figure 6.5a suggests no immediately obvious way to break the structure into particle sub-groups associated with the three peak features in the scattering cross section. Further, due to the complete lack of structure symmetry, the partial cross-section spectra shown in Figure 6.5b indicate that essentially each particle contributes a distinct spectral component although some general trends associated with the features in the complete cross section can be recognized.

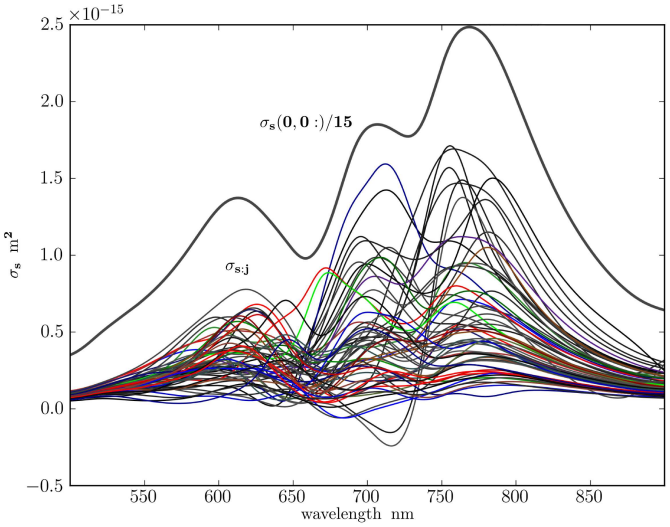
Inspection of the pseudo-color labeled structure shown in Figure 6.6 and of the related images associated with the magnitude of each color channel does allow the three peak features in the complete cross section to be associated with sub-groups of particles within the structure. However these sub-groups are not spatially localized but instead group together particles that share aspects of their local environment. These details are straightforward to evaluate when the pseudo-color labeling is used to indicate the magnitude of each color-channel separately. In particular, particles with more nearest neighbors and more closely-spaced neighbors are associated with the long-wavelength spectral feature (see Figure 6.7); particles with an intermediate number of neighbors are associated with the mid-wavelength feature (see Figure 6.8); and more isolated particles are associated with the short-wavelength feature (see Figure 6.9).

It is also possible using these illustrations to group the more closely-spaced particles into sub-groups containing clusters and short chains of particles (e.g., $\{25, 18, 19, 8, 16, 6\}$, $\{23, 12, 35, 22, 42, 29\}$, $\{9, 21\}$, and $\{30, 24\}$), and this then makes the attribution fully consistent with the special requirements detailed at the

start of the section, that is, that for spectral effects due to constructive superposition, the sub-groups of particles must be sufficiently isolated that inter-group coupling is minimized.



(a) Representation of aggregate structure.



(b) *Complete* and *partial* cross sections (25 nm particles, $\langle a \rangle = 2.1, \sigma_a = 0.2a$).

Figure 6.5: Example of a randomized 61-particle aggregate.

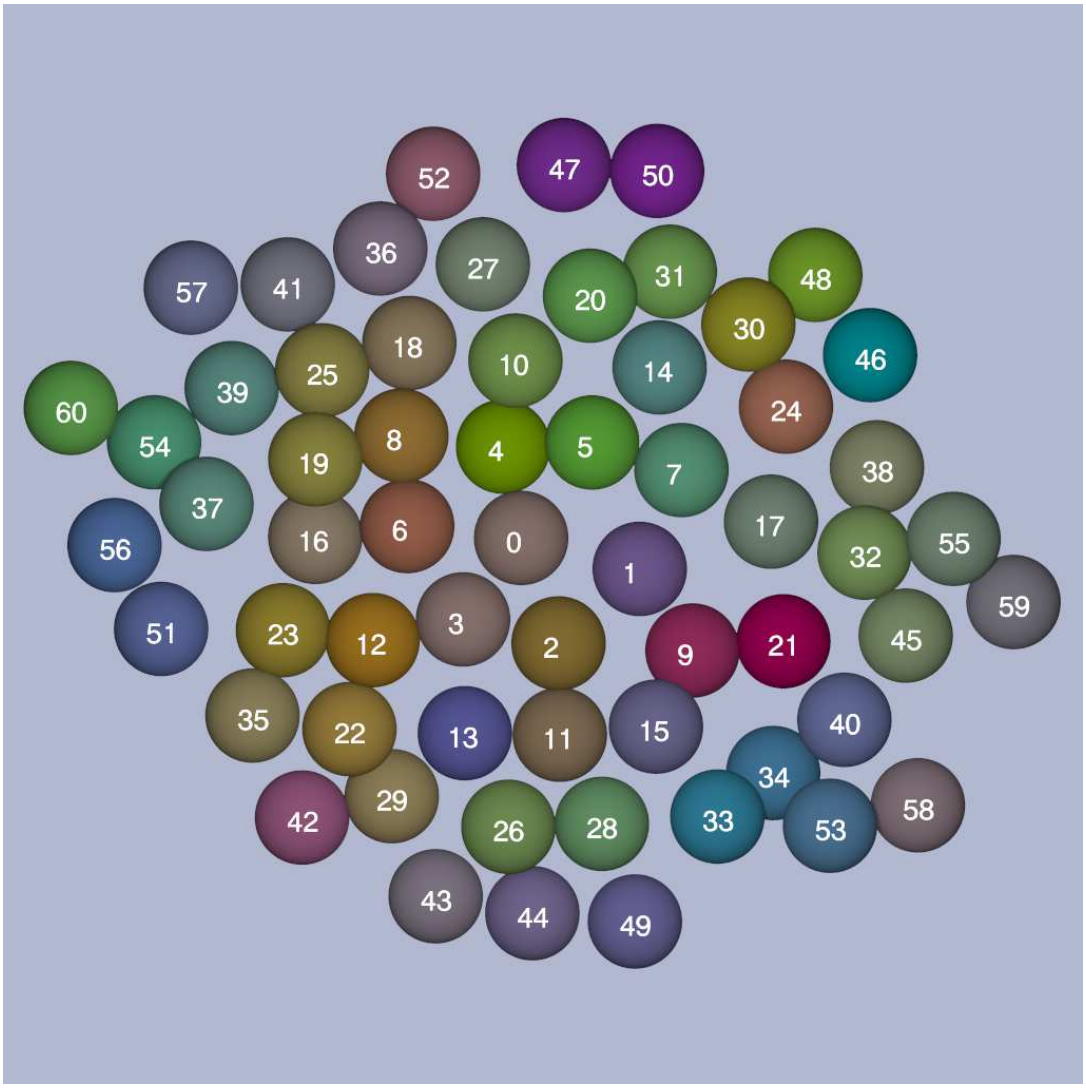


Figure 6.6: Pseudo-color labeling of aggregate structure using colors generated from partial cross-section spectra.

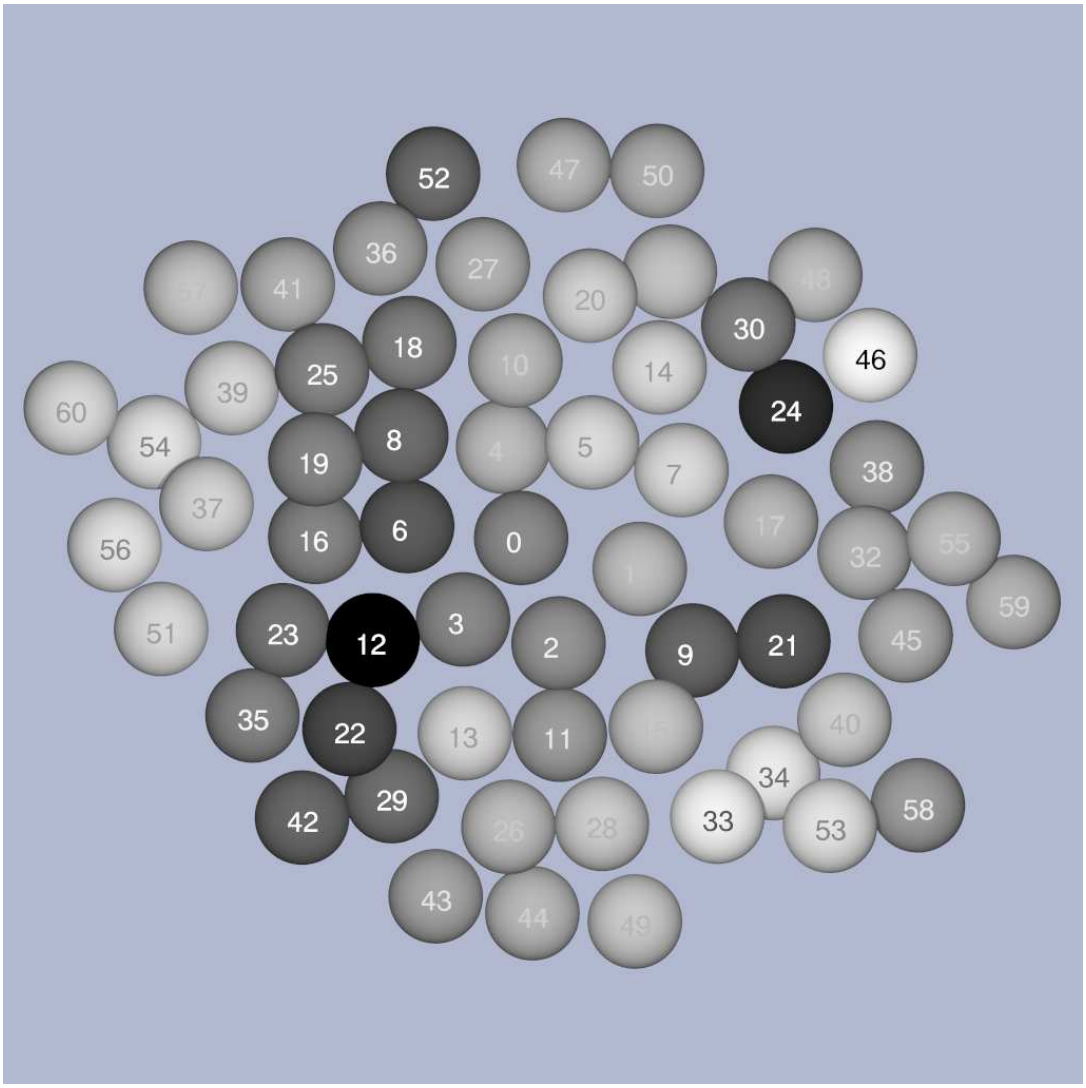


Figure 6.7: Pseudo-color labeling of aggregate structure: magnitude corresponds to long wavelength features (black ↑ magnitude).

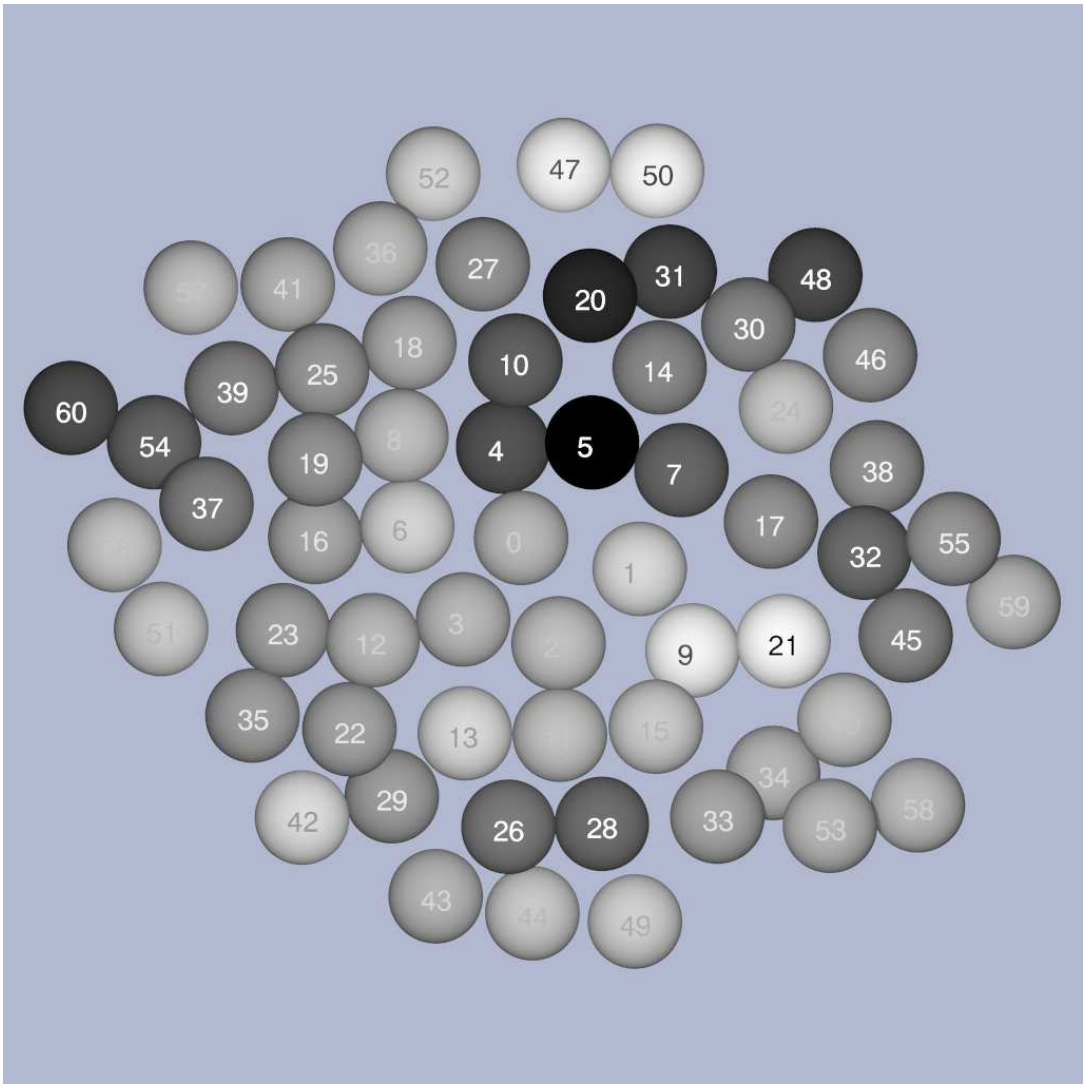


Figure 6.8: Pseudo-color labeling of aggregate structure: magnitude corresponds to intermediate wavelength features (black ↑ magnitude).

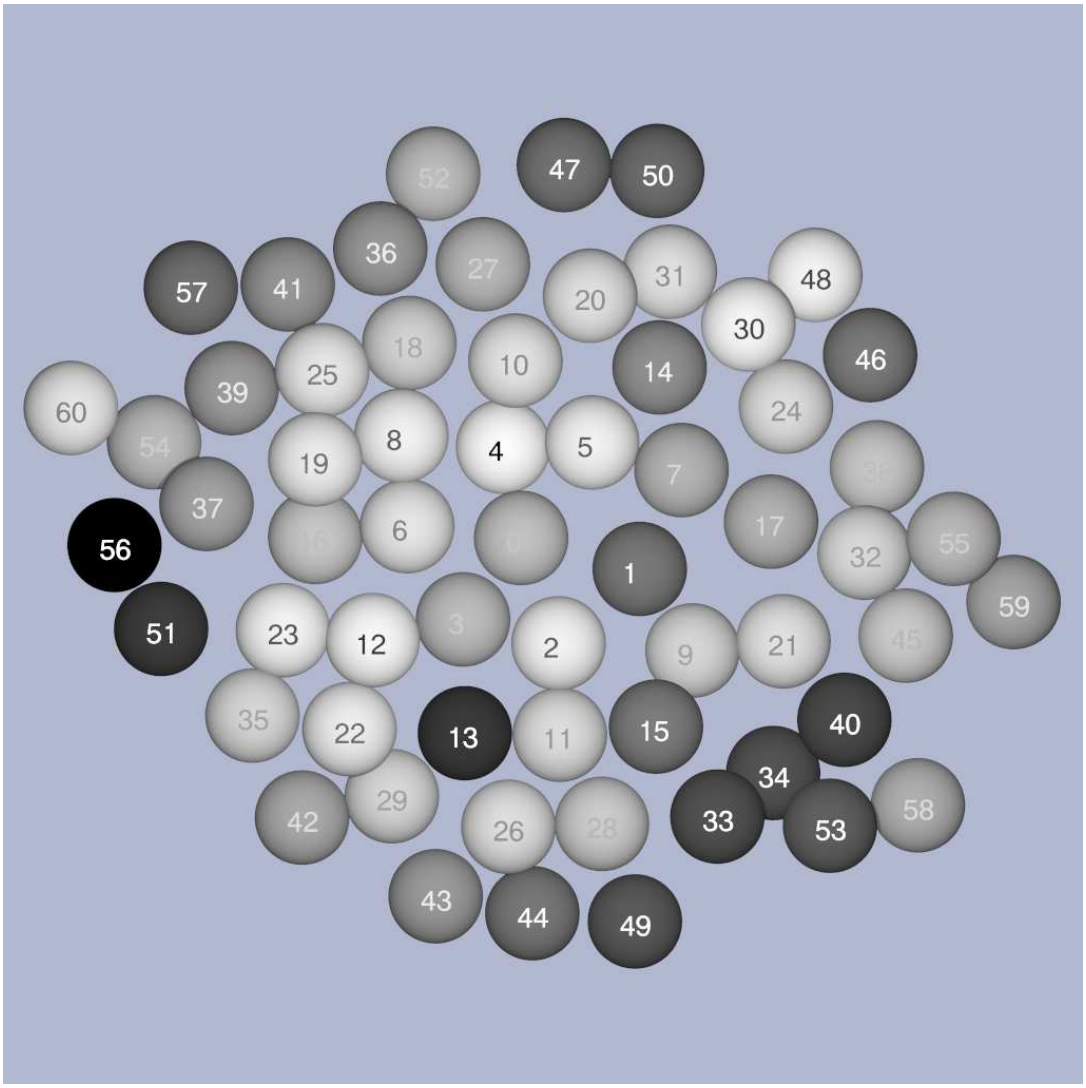


Figure 6.9: Pseudo-color labeling of aggregate structure: magnitude corresponds to short wavelength features (black ↑ magnitude).

6.3.1.3 Implications of constructive superposition

With respect to the cross section of the entire aggregate, spectral features produced by constructive superposition will have only feature scale-lengths comparable to those of the cross sections of the participating sub-groups of particles considered separately. However, as discussed in Chapter 3, for particle groups described by a correlation function with a *uniform* angular aspect, the primary effect of all inter-particle coupling is a red-shift and broadening of cross-section spectral features compared to those of the isolated particles. Although in certain specific cases the peak width of the spectral features may narrow very slightly (for example, as may occur if the imaginary component of the complex dielectric permittivity for the bulk material is reduced in magnitude at the new peak location), this is not the most general trend. Any such peak narrowing is usually overwhelmed by mixing effects from inter-particle coupling from many neighboring particles, which coupling serves to broaden the peaks. In combination these factors lead to the strong conclusion that spectral features produced by constructive superposition can, at the finest level of detail, have only feature scale-lengths comparable to those of the cross sections of the isolated particles.

6.3.2 Destructive superposition

There are many cases where from geometric considerations an aggregate structure can be partitioned into sub-groups but, with respect to near-field coupling between the sub-groups, the minimal inter-group coupling restrictions of Section 6.3.1 do not apply. The general case for this situation is quite complicated, as it is qualitatively equivalent to the case of an aggregate comprised of heterogeneous particles, where inter-particle coupling cannot be neglected. In fact, using the T-matrix formalism and representing the electrodynamics of each sub-group of particles in terms of an

appropriate cluster-centered T-matrix (where in this case the term "cluster" refers to the particle sub-group itself), these two cases can be made formally equivalent. However, a more limited case will be treated here, a case that considers interactions among only a small number of such sub-groups. This will illustrate key aspects of a particular type of spectral feature associated with destructive interference effects.

A resonance that includes contributions from both constructive and destructive interference is known as a *Fano* resonance[120]. This type of resonance was originally analyzed by Rayleigh in 1907[121] in connection with very narrow bright and dark features in the spectra from optical reflection gratings, and the theory was treated comprehensively and put in its modern form by Ugo Fano in 1935 as part of an analysis of the Rydberg spectral lines of the hydrogen atom[122]. The Buetler-Fano formula expresses this type of resonance profile in terms of two parameters, a reduced energy ϵ and an asymmetry parameter q[122, 123]

$$\sigma(\epsilon) = \frac{(\epsilon + q)^2}{\epsilon^2 + 1}, \quad \text{where the reduced energy}$$
$$\epsilon = \frac{2(E - E_0)}{\Gamma},$$

E_0 is the position of the resonance peak, and Γ is the peak width. The Buetler-Fano formula describes many different resonance profiles depending on the value of the asymmetry parameter q, ranging from the symmetric Lorentzian profile through profiles with varying degrees of asymmetry, all the way to a symmetric subtractive *valley* profile. Several examples are shown in Figure 6.10.

Although originally identified in association with optical and quantum-mechanical systems involving interference effects between a continuum transition and a discrete transition, the Fano resonance does not require that these specific types of transition be present. The simplest system manifesting a Fano resonance is comprised of two driven coupled oscillators with damping and with the resonance frequencies

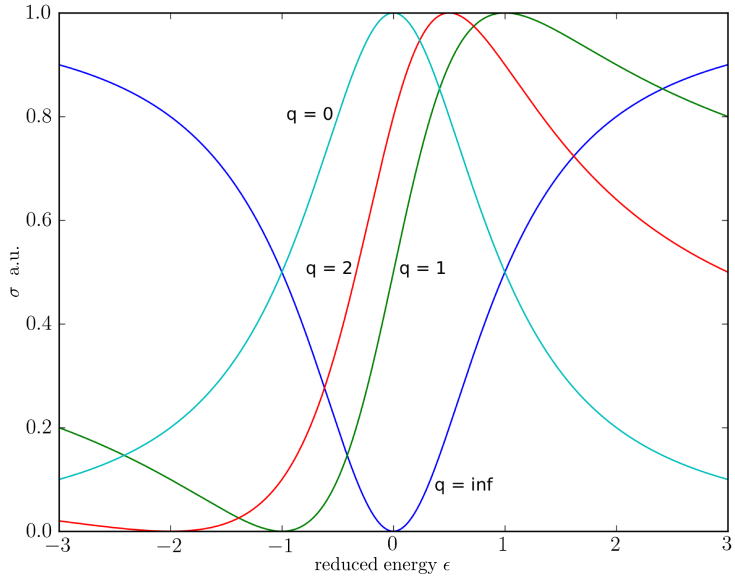


Figure 6.10: Fano resonance profiles with various values for the asymmetry parameter q.

of the two oscillators being sufficiently separated that there is a range of forcing frequencies including frequencies on opposite sides of each oscillator's resonance peak. This allows the existence of frequencies where the responses of the two resonators will be π-radians out of phase. Rapid variations in response amplitude may occur when the interference between the two oscillators changes between constructive and destructive as the driving frequency passes through one of the resonance peaks[124]. This simple Fano-resonance manifesting system enables identification of key system parameters relevant to the presence of Fano resonance in the current context. These are the magnitude and peak-width of the respective resonances, the separation between the resonance peaks, and the degree of coupling between the two resonant systems.

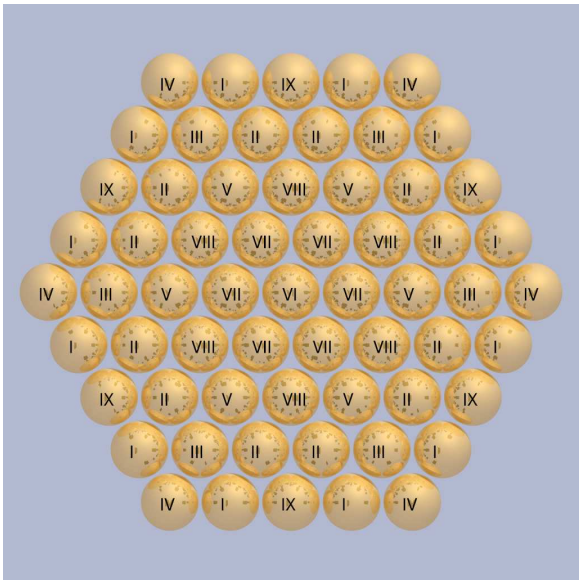
6.3.2.1 An example structure

The aggregate structure represented in Figure 6.11a is a non-randomized version of the 61-particle aggregate analyzed in the previous section (see Figure 6.5a). Examination of the partial scattering cross sections in Figure 6.11b reveals that, due to the symmetry of this structure, each partial cross-section plot is associated with a group of particles. These groups are labeled with Roman numerals. The single sharp negative feature separating the peak in the complete cross section is due to contributions from a set of partial cross sections that rapidly fluctuate over a small range of wavelengths, a set that includes one partial cross section with a subtractive contribution. Such a feature occurring in a partial cross-section analysis is the signature of a Fano resonance.

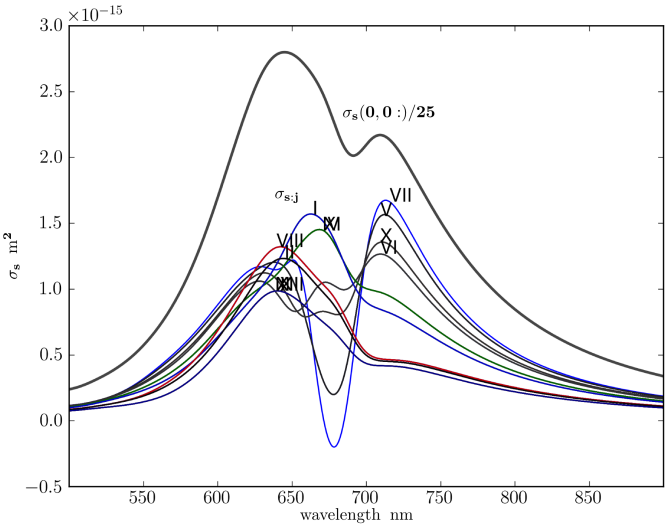
Using a pseudo-color map based on the partial cross-section spectra, Figure 6.12 shows that the sub-group of particles associated with the subtractive peak feature includes the particles at the center of the structure with the most red-shifted partial cross sections, and that the particle at the exact center of the structure is associated with the negative partial cross section. This central sub-group of particles constitutes one of the groups required to form the Fano resonance and, considering the requirement for separation of the resonances of the two participating groups, it seems most likely that the other sub-group required is the one formed by the ring of particles near the periphery of the structure.

The fact that what is evaluated in the partial cross section already includes the mixing among the participating sub-groups makes identification of the sub-groups participating in the Fano resonance somewhat problematic. For example, in the recent literature an analysis of the Fano resonance of a fabricated seven-particle structure shows peak features similar to the present example[116]. In this analysis the authors are able to identify the sub-groups participating in the resonance only by calculating

the full electrodynamic fields of the structure and then identifying which particle is resonating out-of-phase with the others. A comparison is then made between the properties of the full structure and those of a structure with the central particle removed, and this comparison is used to explain details of the interactions among the sub-groups. However, it should be noted that the very coupling between the particle groups responsible for the mixing required to produce the Fano resonance is in many cases also responsible for the separate spectral properties of the sub-groups, and therefore the behavior of the sub-groups as separated from the complete system may have little to do with the specifics of the Fano resonance because these separated sub-groups will generally have completely different resonant properties.



(a) Representation of aggregate structure.



(b) *Complete* and *partial* cross sections (25 nm particles, a = 2.1); labeling of particle *groups* as indicated.

Figure 6.11: Example of regular 61-particle aggregate.

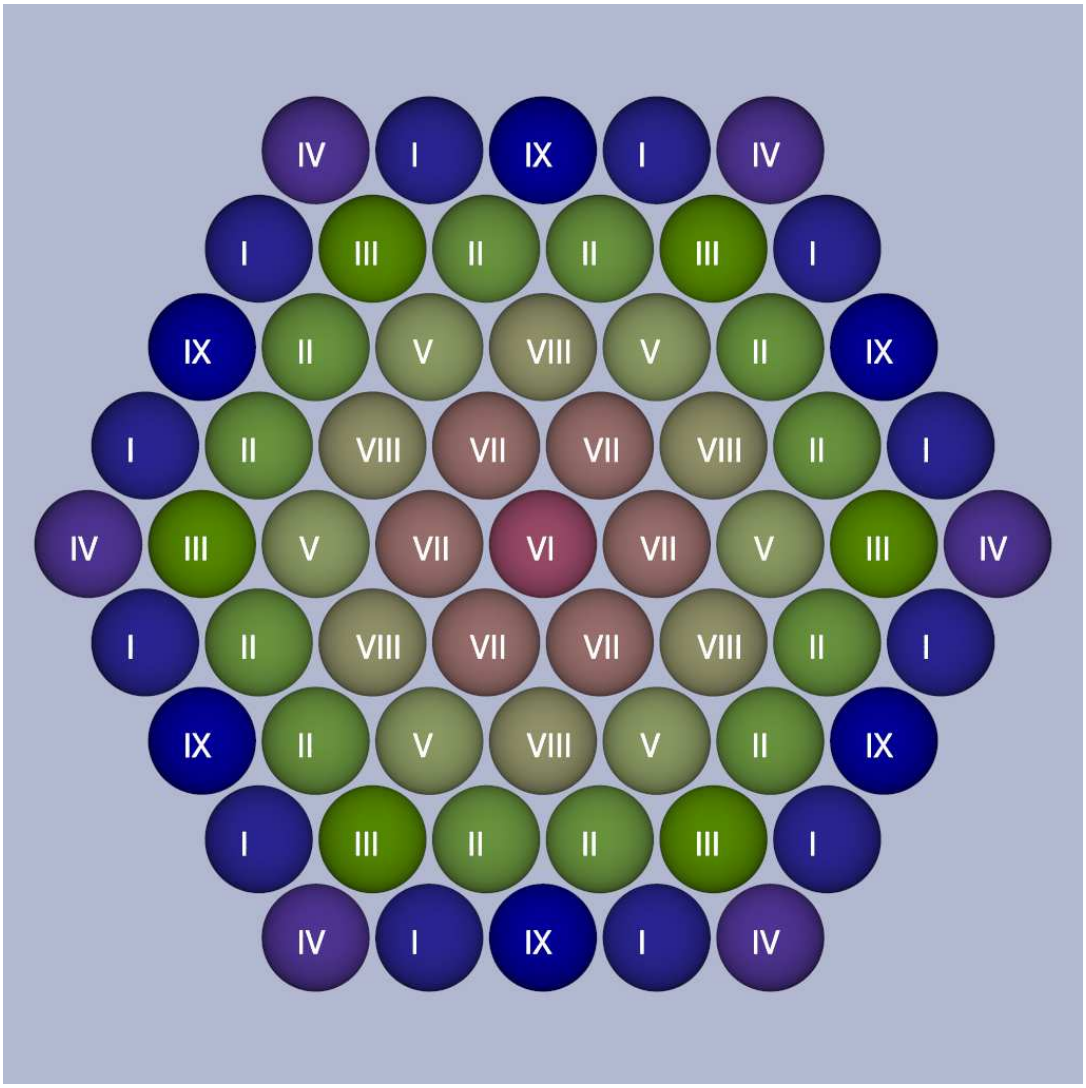


Figure 6.12: Pseudo-color labeling of aggregate system using colors generated from partial cross-section spectra.

6.3.2.2 Statistical effects

A closer comparison between the partial cross-section spectra shown in Figures 6.11b and 6.5b indicates that, although the two systems have the same mean inter-particle spacing, the effects of destructive superposition are greatly reduced in the more random system. Whereas in the regularly spaced system the entire central group of particles participates in the Fano resonance, in the more random system only two particles still participate and, further, the subtractive effect of the destructive superposition is no longer clearly distinguishable in the complete cross section. Randomization has sufficed to distribute the effects of, and thereby to make much less influential, the two sub-groups identified previously as responsible for the destructive resonance. This strongly suggests that, for a general statistical ensemble of aggregates, destructive superposition effects in the associated cross sections will be much less prevalent than constructive effects and, further, that localization of sub-structure may be required in order to produce destructive resonance.

6.3.2.3 Dependence on particle size and inter-particle spacing

The highly sensitive dependence of the Fano resonance on the properties and magnitude of coupling between the resonances of the participating sub-systems implies that features associated with destructive superposition will also depend on any factors affecting inter-particle and inter-group coupling, for example, such factors as inter-particle distance and (see the considerations of Section 6.4 below) particle size. In this regard it is useful to compare cross-section spectra between aggregates where each of these factors is allowed to vary independently although it will be noted that, in general, any changes that affect inter-particle coupling will also affect inter-group coupling. With respect to inter-particle distance, such a comparison is seen in the simulation series of Section 3.3.1 for which it is now possible to portray

the sharp negative feature dividing the peaks as a Fano resonance, which is manifested when inter-particle distance is closer than a certain value and which has a larger effect as particles become more closely spaced. With respect to particle size, as associated with uniformly rescaling the complete aggregate structure, such simulation results are presented in Figure 6.13. This series shows that the very existence of and the magnitude of Fano-resonance features depend strongly on particle size. Worth special notice for this latter simulation series is the fact that, for the largest structure, more than one Fano resonance pattern is evident.

Separately or in combination, this dependence of destructive superposition features on inter-particle distance and particle size can be used as a thresholding effect for the design of experiments, or for the interpretation of experimental results. For example, the presence or absence of Fano resonance features would indicate a significant change in inter-particle spacing, assuming that other influencing factors are held constant.

6.3.2.4 Implications of destructive superposition

With respect to the cross section of the entire aggregate, if spectral features are observed with feature scale-lengths smaller than those of the isolated particle spectra, they must necessarily be associated with destructive superposition. With respect to the aggregate morphology required to produce these features, the aggregates involved must have enough complexity for there to be well-defined sub-groups of particles at small enough inter-group distances to enable sufficient inter-group coupling. Further, preliminary considerations indicate that regional *localization* of these sub-structures may also be required. Interesting possibilities are suggested by these implications, both for the analysis of existing experimental results from live-cell labeling, which may show an increased incidence of very fine spectral features

associated with aggregate complexity, and for the design of new experiments based on thresholding with respect to the presence or absence of destructive resonance features.

6.4 Size dependence, near-field considerations

In Section 3.2.1, the dependence of the dipole component of the near-fields on particle radius R and on distance r from the particle origin was used to argue for the utility of the *radius-normalized* inter-particle distance $a = r/R$ in expressions treating the effects of dipole-dipole coupling. An analogous argument has been used in the literature to suggest a "Universal Scaling" law for cross-section spectral features associated with nanoparticle pairs [119]. However, continuing with analysis using the structure of the preceding section, the simulation series of Figure 6.13 clearly shows that this is an over-simplification and, in fact, may be applicable only when considering structures comprised of the very smallest nanoparticles, if at all.

Using the technique of zeroing out specific multipole orders of the single-particle T-matrices used during the multiple-particle T-matrix calculation, the effect of the various single-particle multipole orders on inter-particle coupling can be evaluated directly. Figure 6.14 shows a comparison of the scattering cross section for the largest structure including inter-particle coupling of all multipole orders and only at the dipole order. In contrast to the analogous collective-mode calculations shown in Figures 6.1 and 6.2, it is clearly evident that, with respect to per-particle interactions, inter-particle coupling induced by higher multipole orders cannot be neglected.

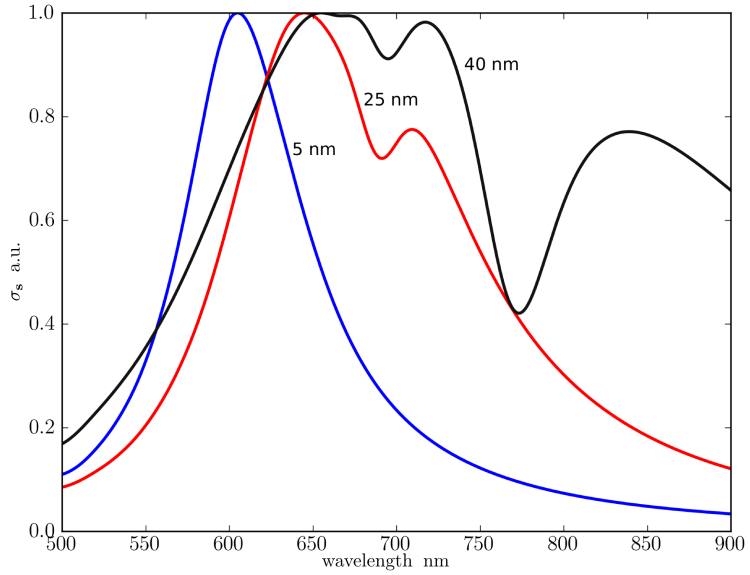


Figure 6.13: Scattering cross sections for the structure of Figure 6.11a (particle sizes as indicated, a = 2.1).

6.4.1 High-order symmetry: implication of the Debye criterion

To evaluate the manner in which these size-dependence effects work, it is useful to examine a calculation of the full electrodynamic fields within the aggregate structure. The result of such a calculation is shown for the square-modulus of the electric field in Figure 6.15. Considering the two-fold symmetry of the dipole fields, it is immediately obvious that contributions from higher-order multipole fields will be required in order to match the boundary conditions at the particle surfaces. In fact, because the largest field contribution present is due to the dipole fields and the reduction in the magnitude of the dipole fields from the particle surface to the surface of an adjacent particle is only about 14% (for the calculation shown, the interparticle spacing is 2.1 times the particle radius), the magnitude of these higher-order

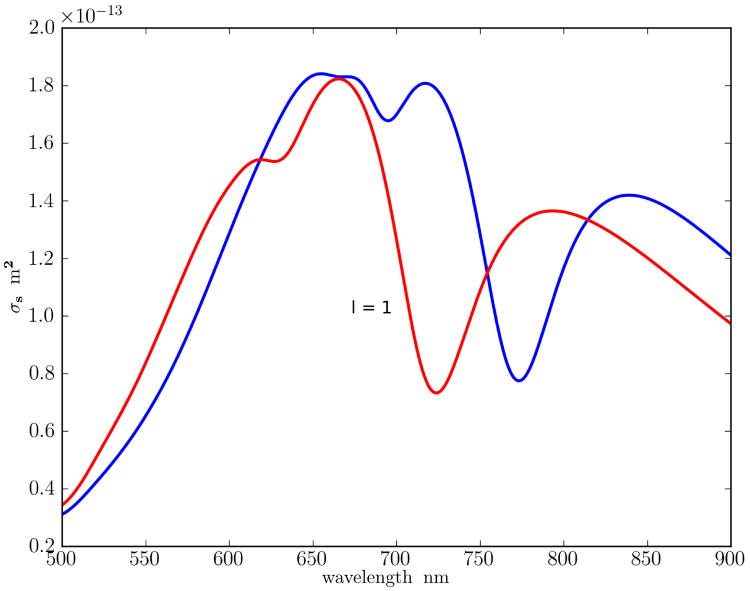
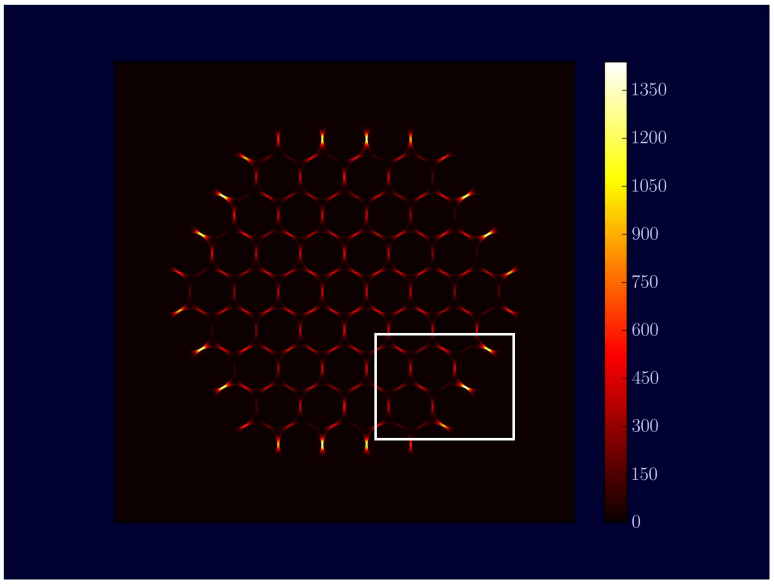
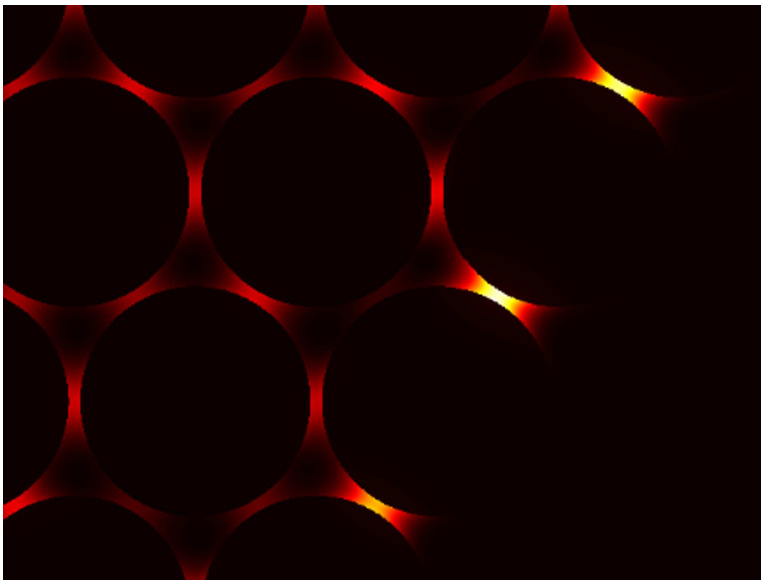


Figure 6.14: Scattering cross sections including all per-particle multipole orders, and only dipole order, for the structure of Figure 6.11a (40 nm particles, a = 2.1)

multipole fields at the particle surfaces must be within the same order as that of the dipole fields. Finally, applying the Debye criterion, since the contribution at each multipole order l scales with respect to the corresponding single-particle T-matrix coefficient, which in turn (when $kr \ll l$) scales exponentially with respect to the particle radius r, these field magnitudes must also scale *exponentially* with respect to particle radius. The conclusion of this analysis is that, unlike the slow variation with size of the scattering spectral features of small single particles, the variation with particle size of scattering spectra from aggregate structures will be much more extreme.



(a) $|\vec{E}|^2$



(b) Enhanced view of indicated ROI.

Figure 6.15: Full-fields calculation for the structure of Figure 6.11a (normalized to incident $|\vec{E}|^2 = 1$; 25 nm particles, a = 2.1).

6.4.2 Limitations of the partial cross-section analysis

Just as an analysis of the collective-mode behavior of the structure omits important details that can only be understood by looking at per-particle interactions, analysis of far-field effects omits important details that can only be understood through consideration of the near-fields. Although demonstrated very directly in the preceding section, this issue requires additional attention with respect to the implications of the partial cross-section analysis which, for the particular type of cross section used, *only* evaluates far-field effects.

With respect to the field square-modulus displayed in Figure 6.15, several observations can be explained using the simple phenomenological considerations of Chapter 3. For example, the fact that the largest field magnitudes occur at the regions of closest approach between neighboring-particle surfaces can be seen as a direct consequence of the radial dependence of the dipole near-fields. The bright regions of inter-particle fields at the periphery of the structure can be explained through consideration of the *cancellation* effect of dipole sources placed side by side rather than along the longitudinal field axis (note that the excitation field for this simulation case has $+$ helical polarization). And, finally, in combination with the partial cross-section analysis of Section 6.3.2.1, the *net* effect of the lack of cancellation for these peripheral particles is *not* an increase in red-shift, which can be explained by the threshold zenith angle analysis of Section 3.2.3.2 (which indicated that the overall spectral effect of any neighboring dipole sources placed at zenith angles less than $\approx 70^\circ$ will be to induce red-shift). All of these observations point toward the more general applicability of the simple rules.

Despite this success of the simple phenomenology, a discrepancy arises when the magnitudes of the partial cross sections are compared to the square-modulus of the near-fields. Figure 6.16 shows a representation of the structure using a color-map

corresponding to the magnitude of the partial scattering cross sections at the same wavelength as that used for the fields calculation. It will be noted that there is no increase in the partial cross-section magnitude corresponding to the increase in the field square-modulus near the peripheral particles.

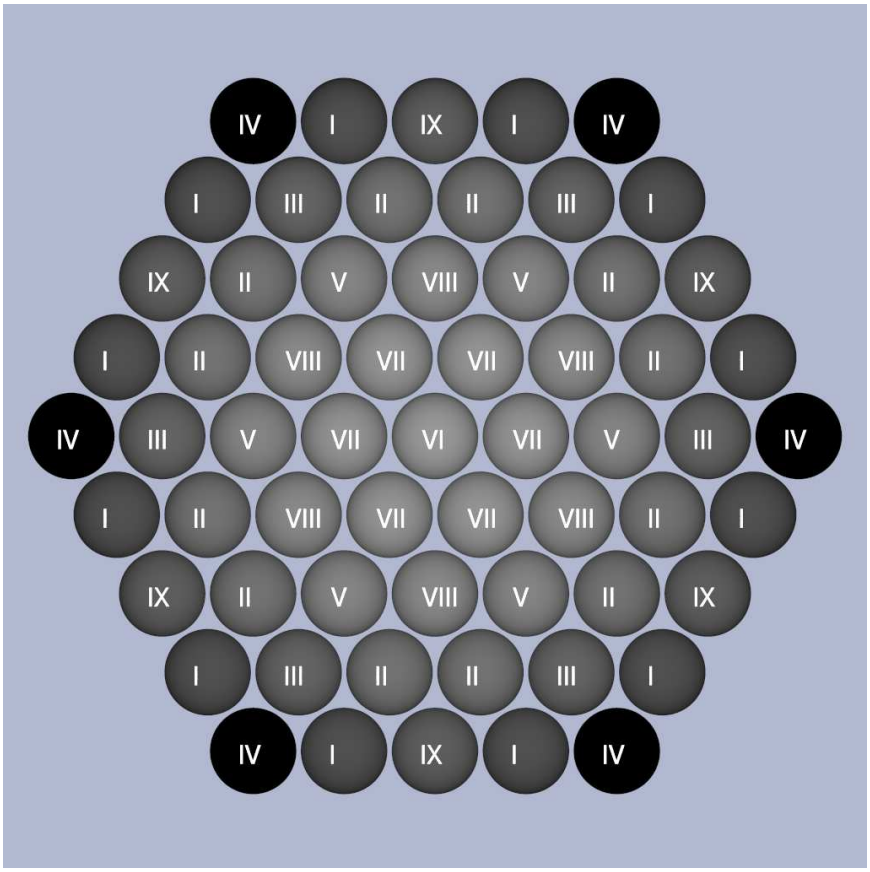


Figure 6.16: Pseudo-color labeling corresponding to partial scattering cross-section magnitudes.

This discrepancy can be explained by using the high-order properties of the VSW fields basis. As the cross-section reductions are evaluated using the far-fields limit of Equations 2.34 and 2.35, any terms in the *electric* VSW $\vec{N}_{lm}$ including the lon-

gitudinal vector spherical harmonic (VSH) $\vec{P}_{lm}(\hat{n})$ do not contribute to this limit and, further, evaluation of the Poynting vector for fields proportional to these terms results only in *circulating* energy flux, which does not contribute to outgoing energy flux in the far-field. An example of a full-fields calculation omitting these non-radiating components is shown in Figure 6.17, and it is now seen that this simulation corresponds quite well with the trends in partial cross-section magnitude shown in Figure 6.16.

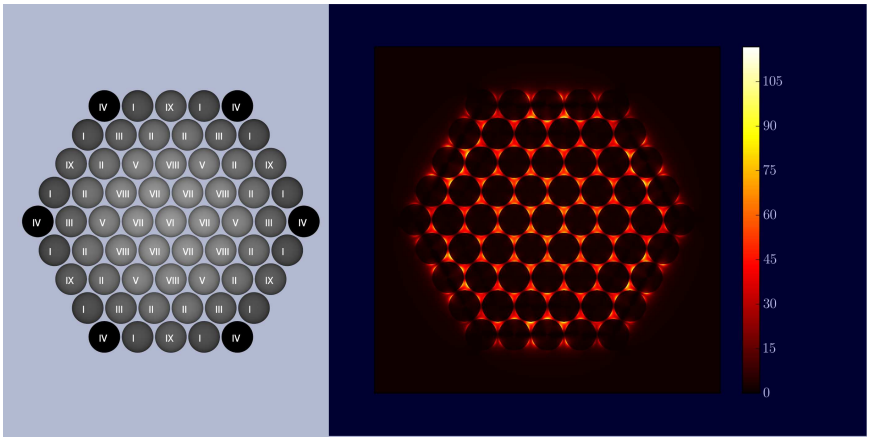


Figure 6.17: Pseudo-color labeling corresponding to partial cross-section magnitudes, and $|\vec{E}|^2$ of *radiating* component of the near-fields
(normalized to incident $|\vec{E}|^2 = 1$; 25 nm particles, a = 2.1).

At this point a promising prospect for future work should be mentioned, in particular, the use of alternative *near-field* formulations of the partial cross section in order to evaluate inter-particle coupling effects. Such expressions would be analogous to more traditional cross-section expressions but would involve both circulating and radiating components of the irradiance at the circumscribing sphere about each individual particle. It appears that this approach will lead to a better technique for determination of the sub-particle groups responsible for destructive superposition effects, as discussed at the end of Section 6.3.2.1, and also will provide insight into

the near-field behavior of inter-particle coupling at higher multipole order.

6.5 Summary

The analysis of the present chapter has focused on aspects of the electrodynamics of aggregate structures affecting spectral features in the optical cross sections. This treatment involved application of the analytic properties of the multiple-particle T-matrix technique at several different levels, proceeding from an examination of collective-mode effects, through an examination of per-particle effects, to an examination of effects associated with details of the single-particle T-matrices. At this third level, additional considerations based on calculations of the full electrodynamic fields were used to amplify several key points of the analysis.

For aggregate structures with sizes corresponding to those produced in live-cell labeling experiments, it was shown that collective-mode effects depending on higher multipole orders can largely be neglected. Care must be taken when relating these results to those in the literature because the latter often involve the analysis of much larger structures [116, 117, 118, 119]. For the dipole collective mode, properties of the polarized cross sections were consistent with the general trends predicted in the discussion in Chapter 3 of effects related to aggregate aspect ratio. However, the fact that fine details appear in the peak structure for the dipole-only cross sections indicates that these details can only be associated with per-particle effects.

Per-particle effects were evaluated by considering two different types of partitioning of the aggregate into sub-groups of particles. The first type treated sub-groups with only minimal inter-group coupling and was shown to be associated with constructive superposition features in the cross-section spectra, which features can in general only have a smallest feature scale-length at the same order as that of the

isolated particle cross sections. An important aspect of this first type of partitioning was seen to be that these sub-groups of particles are not necessarily region-localized within the structure but in many cases are comprised of particles that are distributed throughout the structure but that are similar in some way involving their environments, for example, similar with respect to the number and spacing of neighboring particles.

The second type of partitioning involved sub-groups of particles with more significant inter-group coupling. This type of partitioning requires, first, that the sub-groups themselves be well defined by manifesting significant inter-particle coupling among particles within the group and, second, that the details of the inter-group geometry are such that sufficient coupling also exists among the particle groups themselves. There is some question as to whether this type of partitioning may also be associated with particle sub-groups not region-localized, but more work needs to be done before a definite answer is available. Although this second type of partitioning can also be associated with constructive superposition features in the cross-section spectra, the key difference from the first type is the possibility of destructive superposition features. These destructive features, in addition to having a subtractive effect, can also be associated with much finer feature scale-length.

With respect to these two types of particle sub-groups within an aggregate, it is important to note that a given aggregate may simultaneously contain both types of sub-group.

Finally, a closer look was taken at size dependence of cross-section features and it was shown that, in contrast to the predictions of the simple phenomenology of Chapter 3 and assertions in the literature[119], there can in general be no universal scaling based on dipole-dipole coupling for cross-section features with respect to aggregate size. This characteristic of scaling behavior is explained by analyzing the

implications of the Debye criterion in terms of the single-particle T-matrix coefficients and noting the distinct difference between the influence of these coefficients in the near-field case and their influence in the far-field case. With respect to the partial cross-section analysis applied in many of these sections, it was noted that it only includes far-field effects and, therefore, that more work needs to be done to elaborate the details of these higher multipole-order near-field interactions.

Chapter 7

Conclusion

The work presented in this dissertation has applied tools from classical and computational electrodynamics toward enhancing understanding of the optical properties associated with a particular real-world system, viz., aggregates produced during the labeling of biological cells and tissue with plasmonic nanoparticles. As with many applications of physics to complex systems, the objective was not the more traditional one of the *calculation* of physical results, rather, the objective was the development of a set of rules allowing *interpretation* of the general, qualitative behavior of a class of systems. For such an application, detailed information may only be available in a statistical sense, and this information then functions largely as a set of constraints on system parameters.

In the first part of the dissertation, from considerations of the distance dependence and the angular symmetry of the dipole component of the near-fields induced between adjacent nanoparticles, a comparative approach is used to develop a set of simple rules governing the trends in spectral shift of the optical cross sections for these aggregate structures.

For particles with radius sufficiently less than about $1/k$ (~ 100nm), the polarizability is proportional to the *volume* of the particle and this, in combination with the $1/r^3$ dependence of the dipole component of the near-fields, leads to a similarity in the behavior of dipole-dipole coupling with respect to particle size. In particular, for inter-particle surface-to-surface distances less than the order of the particle radius,

the near-fields induce measurable changes in the cross-section spectra. At larger inter-particle distances, although inter-particle coupling can be neglected, across the aggregate structure as a whole the phase-shift of the fields is minimal, and thus the fields must still be added together *coherently*. This leads to a quadratic dependence of the magnitude of the scattering cross section on the number of particles in the aggregate. Additionally, even at closer inter-particle distances, this quadratic dependence will still apply with respect to the wavelength-integrated scattering cross section.

Although the dipole field is not symmetric with respect to angular orientation, the fact that, with respect to changes in angle, the magnitude of the field at its maximum is twice that at its minimum is sufficient to ensure that, when particles are added uniformly to an aggregate, spectral shift due to inter-particle coupling will always occur. This is equivalent to the observation that, all else being equal, aggregates containing more particles will have cross sections with more spectral shift. However, additional consideration of the effect, on particles in its interior, of adding particles to the periphery of the aggregate structure leads to the observation that this spectral shift due to added particles must eventually saturate. For an aggregate larger than a certain size, the details of which depend on the dimensionality and on the particle-packing distribution of the aggregate's structure, adding more particles will not result in additional spectral shift.

Despite the preceding, the angular properties of the dipole field do affect the spectral shift for structures with asymmetric aspect ratio. In general, it can be asserted that elongated structures will manifest additional spectral shift compared to less elongated structures, but the caveat must be added that the aspect ratio corresponding to the maximum degree of spectral shift is somewhat less than that corresponding to maximum elongation.

In combination, these simple rules were also shown to be able to predict the optical properties of more random aggregate structures with large numbers of particles. In particular, given a conserved total volume for the structure and provided the angular aspect of the correlation function describing the particle positions is *uniform*, the effect of any non-uniformity with respect to the *radial* aspect of the correlation function will always be to increase the inter-particle coupling. At the other extreme, for different correlation functions with equivalent mean inter-particle distance, when the angular aspect of the correlation function produces more chain-like particle packing, increased inter-particle coupling results and, when it produces more plate-like packing, decreased inter-particle coupling results.

After identification of this small number of statistical parameters that qualitatively determine the optical properties of the cross sections, the possibility that more *quantitative* predictions could also be made was investigated. A numerical experiment was designed to examine whether key parameter values required in low-order approximation formulas could be obtained either through numerical computation or through analysis of experimental observations. The answer is that, for suitably restricted ranges of values, low-order formulas do exist and will be usable in experimental contexts.

Although these simple phenomenological rules are useful both for the design of new experiments and for qualitative interpretation of many experimental measurements, they are insufficient to explain complicated peak structures experimentally observed. Further, these rules alone cannot predict the range of possible optical properties that might be available from aggregate structures. Such properties may be associated with the assembly of aggregate structures in response to motion of attached target biomolecules during biological labeling experiments but, also of great interest, are potential properties of structures deliberately fabricated for specific

purposes such as application to biomolecular sensing or to multiplexed labeling. Addressing these deficiencies required an additional approach.

The additional approach undertaken positions itself midway between a purely theoretical treatment and a completely computational treatment and, from this perspective, several new insights emerge. Specifically, rather than following the approach of most modern-day computational electrodynamics, which approach represents the electrodynamic system using as *compact* a basis for field representation as possible, a contrasting approach has been taken, one using a very high-order (in fact, a *redundant*) basis representation. The computational approach used is that of the multiple-particle T-matrix technique. Through the use of this technique information is provided about the collective electrodynamic modes of the system, and in addition about the electrodynamic modes associated with the individual particles as well as with groups of particles.

The elaboration of the multiple-particle T-matrix technique was presented in the context of the VSW basis and its associated translation operator, thereby enabling formal construction of the multiple-particle T-matrix from the single-particle T-matrices representing the component particles. The fact that this computational approach has heretofore not been widely used to its full capability led to the need for development of a completely new computational implementation, and this aspect of the work was detailed in the next chapter. Because this numerical code-base was developed in response to the requirements of several different biophysical projects, the code-base constitutes a most general set of tools potentially useful over a wide range of applications.

In the concluding sections of the dissertation, the unique perspective afforded by the multiple-particle T-matrix technique was used to continue elaboration of the phenomenological description of aggregate electrodynamics. First of all, an analysis of

the separate contribution from each collective multipole order, to the optical cross sections of example structures, was used to show that the experimentally observed multiple peaks in cross-section spectra are not associated with distinct collective modes but in fact can only be attributed to the per-particle modes of these structures. This association of cross-section features with *internal* degrees of freedom is conceptually distinct from the general electrodynamic behavior of single-particle systems comprised of *uniform* dielectric media separated by well-defined boundary surfaces with no surface currents. A possible implication of this is that the electrodynamics of these aggregate structures will not be adequately described by effective-medium approximations. (This possible implication needs to be investigated in more detail.)

Next, analysis of the per-particle contributions to the complete cross section were used to associate cross-section spectral features with different types of partitioning of the aggregate into sub-groups of particles. The magnitude of inter-group coupling was used to distinguish between two broad classes of partitioning, each associated with a particular type of spectral feature in the cross section.

The first type of partitioning treated sub-groups with minimal inter-group coupling and was shown to be associated with constructive superposition features in the cross-section spectra, which features can in general only have a smallest feature scale-length at the same order as that of the isolated particle cross sections. An important aspect of this first type of partitioning was that these sub-groups of particles are not necessarily region-localized within the structure but in many cases are comprised of particles distributed throughout the structure and similar in some way with respect to their local environments, for example, with respect to the number and spacing of neighboring particles.

The second type of partitioning involved sub-groups of particles with more sig-

nificant inter-group coupling. This type of partitioning requires, first, that the sub-groups themselves be well defined by manifesting significant inter-particle coupling among particles within the group and, second, that the details of the inter-group geometry are such that sufficient coupling also exists among the particle groups themselves. Although this second type of partitioning can also be associated with constructive superposition features in the cross-section spectra, the key difference from the first type is the possibility of destructive superposition features. These destructive features, in addition to having a subtractive effect, can also be associated with much finer feature scale-length.

Finally, a closer look was taken at size dependence of cross-section features and it was shown that, in contrast to the predictions of the simple phenomenology of Chapter 3 and assertions in the literature[119], there can in general be no universal scaling based on dipole-dipole coupling for cross-section features with respect to aggregate size. This characteristic of scaling behavior is explained by analyzing the implications of the Debye criterion in terms of the single-particle T-matrix coefficients and noting the distinct difference between the influence of these coefficients in the near-field case and their influence in the far-field case. With respect to the partial cross-section analysis applied in many of these sections, it was noted that it includes only far-field effects and, therefore, that more work needs to be done to elaborate the details of these higher multipole-order near-field interactions. It is concluded that in many cases an alternative *near-field* cross-section expression will be useful for implementing computationally efficient evaluation of near-field effects, and for elaborating the relationship between near-field and far-field electrodynamics.

This in-depth analysis associating cross-section spectral features with the specifics of aggregate morphology allows new interpretation of existing experimental results.

In particular, the association of dark-mode peak splitting with an aggregate-size threshold will enable these cross-section features to be used for identifying aggregates containing more than a specified number of particles or, equivalently, aggregates with mean inter-particle coupling above a certain threshold. During actual application the details of this technique need to be carefully calibrated on a case-by-case basis.

Especially noteworthy about all of these analyses is that neither the isolated collective-mode contributions nor the per-particle contributions to cross-section features can be directly evaluated through physical measurement or, for that matter, through the most widely used computational electrodynamic techniques.

It is expected that the analytic approaches demonstrated, as well as the computational tool-set developed, during the course of this work will find wide application. They can, for example, be applied to cost-effective, high-speed, computer-based optimization of the design of aggregate structures with specific properties, like multi-peak cross sections for multiplexed labeling, or like specific near-field configurations to be applied to surface-enhanced Raman scattering. Or they can, for another example, be applied to direct comparison of T-matrix-based and radiative-transport-based calculations of light propagation in biological tissue. And there are potential applications in areas of physics other than electrodynamics that use an analogous multipole-fields basis, for example, fluid dynamics.

Work remains to be done on the theory itself. The work presented here has focused primarily on understanding qualitative features of the spectral properties of the scattering cross section and on understanding the factors affecting the *total* cross section. Issues of cross-section *angular* symmetry have been addressed only minimally. Symmetry will be important for applications using larger particles with significant contributions from higher multipole terms, and for applications where

scattering is collected from non-statistically-averaged single particles and aggregates. Finally, further investigation is needed of the relationship between near-field and far-field behavior and, analogously, between per-particle and collective-mode behavior.

Appendix 1

Abridged documentation of numerical codes

1.0.1 Overview of the appendix

This appendix constitutes an abridged reference manual for the T-matrix code-base. Its purposes are to provide a top-level description of the code-base's object-oriented design and implementation, to present a useful inventory of the code-base's more important capabilities, and to serve as a document of record.

Initially it was planned to have the appendix include a complete reference manual, but this has been ruled out by the fact that the manual has turned out to contain over a thousand pages. It was felt that such a large manual would have for readers less value than an abridged version presenting the most important features of the code-base's design and implementation but suppressing less important aspects.

Clearly, the most important features of the code-base are those directly associated with the T-matrix problem-space:

- the representation of the VSW basis, and the associated translation operator;
- efficient and accurate evaluation of the VSW basis on arbitrary grids;
- a generalized and expandable implementation of the single-particle T-matrix algorithm;
- multiple implementations of the multiple-particle T-matrix algorithm;

- full-fields and cross-section calculation from single-particle, cluster-centered, and multi-centered T-matrix representations.

Other features, definitely important but not directly associated with the T-matrix problem-space, are:

- number, functor, and linear-algebra-related interfaces for the code-base's handling of arbitrary-precision numbers;
- classes and methods supporting a transparent interface to thread-level and process-level parallelism.

The following features, though completely implemented and some of which are required in order for the code-base to function, have largely been omitted from the abridged reference manual:

- an alternative implementation of the VSW translation operator;
- various extensions to the `gmm` template library, including a fully-scalable out-of-core block-matrix implementation, singular-value decomposition, and QR-factoring algorithms;
- a convenient and robust interface to high-level scripting languages, including an MPI-interface for coarse-grained scripting of MPI applications.

The analysis and interface-design presented in this appendix, including any code fragments, are released here under the “GNU Lesser General Public License”, with the standard disclaimer:

```
Program and program fragments.

          Copyright (C) 2010 Kort Travis.
          All rights reserved.

This program is free software; you can redistribute
it and/or modify it under the terms of the GNU Lesser
General Public License as published by the Free
Software Foundation; version 2.1 of the License.

This program is distributed in the hope that it will
be useful, but WITHOUT ANY WARRANTY; without even the
implied warranty of MERCHANTABILITY or FITNESS FOR A
PARTICULAR PURPOSE. See the GNU Lesser General Public
License for more details.

You should have received a copy of the GNU Lesser
General Public License along with this program; if not,
write to the Free Software Foundation, Inc., 59 Temple
Place - Suite 330, Boston, MA 02111-1307, USA.
```

This reference manual was generated from the source code itself, which code has embedded within it comments specially-formatted to enable such generation by the open-source tool `doxygen`[125].

1.1 Module Index

1.1.1 Modules

Here is a list of all modules:

1.2 Namespace Index

1.2.1 Namespace List

Here is a list of all documented namespaces with brief descriptions:

1.3 Class Index

1.3.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

1.4 Class Index

1.4.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

1.5 Module Documentation

1.5.1 arbitrary precision

Support for fully-templated, arbitrary precision linear algebra.

Namespaces

- namespace linalg::sparse

  Support for sparse and block-sparse vectors and matrices.

- namespace mere

  Unified, minimalist, and thread-safe, arbitrary precision integer, real, and complex number classes.

- namespace linalg

  Support for fully-templated, arbitrary precision linear algebra.

- namespace number

  Interface traits and primitive functions for POD and arbitrary-precision number classes.

- namespace numerical_functor

  Classes and methods supporting object-oriented calculations of mathematical special functions.

1.5.1.1 Detailed Description

Support for fully-templated, arbitrary precision linear algebra.

1.5.2 T-matrix

Single and multiple-particle T-matrix calculation.

Namespaces

- namespace geometry

  Region primitives and functions related to boundary-surface and lattice calculations.

- namespace TMatrix

  Classes and utility methods specific to single and multiple-particle T-matrix calculation.

1.5.2.1 Detailed Description

Single and multiple-particle T-matrix calculation.

1.5.3 parallel programming

Support classes and methods for object-oriented multi-thread and multi-process coding.

Namespaces

- namespace commUtil

  Unified low-level binary interface to disk-resident, memory-resident, and MPI inter-process data transfer.

- namespace distributed_solver_module

  Stand-alone interface to Trilinos Belos solvers: separates Trilinos classes from numerical_functor namespace.

- namespace parallelUtil

  Support classes and methods for object-oriented multi-thread and multi-process coding.

1.5.3.1 Detailed Description

Support classes and methods for object-oriented multi-thread and multi-process coding.

1.6 Namespace Documentation

1.6.1 commUtil Namespace Reference

Unified low-level binary interface to disk-resident, memory-resident, and MPI inter-process data transfer.

Classes

- class comm_error

  Base-class for binary-I/O exceptions.

- class abstractCommHandle

  An abstract base class to unify functionality of binary-I/O streams. These classes include: file-streams, MPI::communicator-streams, and memory I/O-streams.

- class fileHandle
- class memoryCommHandle
- class processCommHandle

Functions

- template<class T >

  bool **writeBinary** (abstractCommHandle *fp, const linalg::abstractMatrix< T > &a)

- template<class T >

  bool **readBinary** (abstractCommHandle *fp, linalg::abstractMatrix< T > &a)

- template<class T >

  bool **writeBinary** (abstractCommHandle *fp, const TMatrix::multiCentered_tmatrix< T > &t)
- template<class T >

  bool **readBinary** (abstractCommHandle *fp, TMatrix::multiCentered_-tmatrix< T > &t)
- template<class T >

  bool **writeBinary** (abstractCommHandle *fp, const TMatrix::clusterCentered_tmatrix< T > &t)
- template<class T >

  bool **readBinary** (abstractCommHandle *fp, TMatrix::clusterCentered_-tmatrix< T > &t)
- size_t **read** (void *ptr, size_t size, size_t nitems, abstractCommHandle *h) throw ()
- size_t **write** (const void *ptr, size_t size, size_t nitems, abstractCommHandle *h) throw ()
- void **close** (abstractCommHandle *&h) throw ()
- abstractCommHandle * **open** (const char *filename, const char *mode) throw ()
- abstractCommHandle * **open** (const char *pathname, int flags, mode_-t mode) throw ()
- void **seek** (abstractCommHandle *fp, long offset, int whence) throw ()
- long **tell** (abstractCommHandle *fp) throw ()
- template<class U >

  void **gather** (const U &u, abstractCommHandle *h) throw ()

- template<class U >
  void **gather** (const U &u, std::vector< U > &vU, abstractCommHandle *h) throw ()
- template<class U >
  void **all_gather** (const U &u, std::vector< U > &vU, abstractCommHandle *h) throw ()
- template<class U >
  void **scatter** (U &u, abstractCommHandle *h) throw ()
- template<class U >
  void **scatter** (const std::vector< U > &vU, U &u, abstractCommHandle *h) throw ()
- template<class U >
  bool **writeBinary** (abstractCommHandle *fp, const U &u)
- template<class U >
  bool **readBinary** (abstractCommHandle *fp, U &u)
- template<class U >
  size_t **binarySize** (const U &u)
- bool **readBinary** (abstractCommHandle *h, std::string &s) throw ()
- bool **writeBinary** (abstractCommHandle *h, const std::string &s) throw ()
- size_t **binarySize** (const std::string &s) throw ()
- template<class T >
  bool **writeBinary** (abstractCommHandle *fp, const TMatrix::factoredPropagator< T > &p)
- template<class T >
  bool **readBinary** (abstractCommHandle *fp, TMatrix::factoredPropagator< T > &p)

- template<typename index_type_ , size_t N>

  std::ostream & **operator**<< (std::ostream &os, const linalg::sparse::nDimensionalKey< index_type_, N > &key)
- template<typename VECT , size_t DIM, size_t NDIM>

  std::ostream & **operator**<< (std::ostream &o, const linalg::sparse::sparseSubdimensionReference< VECT, DIM, NDIM > &v)
- template<typename T >

  std::ostream & **operator**<< (std::ostream &o, const linalg::sparse::indefiniteVector< T > &v)
- template<typename T >

  std::ostream & **operator**<< (std::ostream &o, const linalg::sparse::indefiniteSparseMatrix< T > &m)
- template<class T >

  bool **writeBinary** (abstractCommHandle *fp, const std::vector< T > &V)
- template<class T >

  bool **readBinary** (abstractCommHandle *fp, std::vector< T > &V)
- template<class T >

  size_t **binarySize** (const std::vector< T > &v)
- template<class T >

  bool **writeBinary** (abstractCommHandle *fp, const std::list< T > &U)
- template<class T >

  bool **readBinary** (abstractCommHandle *fp, std::list< T > &U)
- template<class T >

  size_t **binarySize** (const std::list< T > &u)

- template<>

  bool **writeBinary**< **double** > (abstractCommHandle *fp, const std::vector< double > &V)
- template<>

  bool **readBinary**< **double** > (abstractCommHandle *fp, std::vector< double > &V)
- template<>

  bool **writeBinary**< **std::complex**< **double** > > (abstractCommHandle *fp, const std::vector< std::complex< double > > &V)
- template<>

  bool **readBinary**< **std::complex**< **double** > > (abstractCommHandle *fp, std::vector< std::complex< double > > &V)
- template<>

  bool **writeBinary**< **long** > (abstractCommHandle *fp, const std::vector< long > &V)
- template<>

  bool **readBinary**< **long** > (abstractCommHandle *fp, std::vector< long > &V)
- template<>

  bool **writeBinary**< **size_t** > (abstractCommHandle *fp, const std::vector< size_t > &V)
- template<>

  bool **readBinary**< **size_t** > (abstractCommHandle *fp, std::vector< size_t > &V)
- template<class T >

  bool **writeBinary** (abstractCommHandle *fp, const gmm::dense_matrix< T

> &M)

- template<class T >

  bool **readBinary** (abstractCommHandle *fp, gmm::dense_matrix< T > &M)

- template<class T >

  size_t **binarySize** (const gmm::dense_matrix< T > &M)

- template<class T >

  bool **writeBinary** (abstractCommHandle *fp, const gmm::row_matrix< gmm::slvector< T > > &M)

- template<class T >

  bool **readBinary** (abstractCommHandle *fp, gmm::row_matrix< gmm::slvector< T > > &M)

- template<class T >

  size_t **binarySize** (const gmm::row_matrix< gmm::slvector< T > > &M)

- template<>

  bool **writeBinary**< **double** > (abstractCommHandle *fp, const gmm::dense_matrix< double > &M)

- template<>

  bool **readBinary**< **double** > (abstractCommHandle *fp, gmm::dense_-matrix< double > &M)

- template<>

  bool **writeBinary**< **std::complex**< **double** > > (abstractCommHandle *fp, const gmm::dense_matrix< std::complex< double > > &M)

- template<>

  bool **readBinary**< **std::complex**< **double** > > (abstractCommHandle *fp, gmm::dense_matrix< std::complex< double > > &M)

- template<class T >

  bool **writeBinary** (abstractCommHandle *fp, const gmm::slvector< T > &V)

- template<class T >

  bool **readBinary** (abstractCommHandle *fp, gmm::slvector< T > &V)

- template<class T >

  size_t **binarySize** (const gmm::slvector< T > &V)

- template<>

  bool **writeBinary**< **double** > (abstractCommHandle *fp, const gmm::slvector< double > &V)

- template<>

  bool **readBinary**< **double** > (abstractCommHandle *fp, gmm::slvector< double > &V)

- template<>

  bool **writeBinary**< **std::complex**< **double** > > (abstractCommHandle *fp, const gmm::slvector< std::complex< double > > &V)

- template<>

  bool **readBinary**< **std::complex**< **double** > > (abstractCommHandle *fp, gmm::slvector< std::complex< double > > &V)

- template<>

  bool **writeBinary**< **double** > (abstractCommHandle *fp, const gmm::row_-matrix< gmm::slvector< double > > &M)

- template<>

  bool **readBinary**< **double** > (abstractCommHandle *fp, gmm::row_-matrix< gmm::slvector< double > > &M)

- template<>
  size_t **binarySize**< **double** > (const gmm::row_matrix< gmm::slvector< double > > &M)
- template<>
  bool **writeBinary**< **std::complex**< **double** > > (abstractCommHandle *fp, const gmm::row_matrix< gmm::slvector< std::complex< double > > > &M)
- template<>
  bool **readBinary**< **std::complex**< **double** > > (abstractCommHandle *fp, gmm::row_matrix< gmm::slvector< std::complex< double > > > &M)
- template<>
  size_t **binarySize**< **std::complex**< **double** > > (const gmm::row_matrix< gmm::slvector< std::complex< double > > > &M)
- template<class KEY , class DATA >
  bool **writeBinary** (abstractCommHandle *fp, const std::map< KEY, DATA > &M)
- template<class KEY , class DATA >
  bool **readBinary** (abstractCommHandle *fp, std::map< KEY, DATA > &M)
- template<class KEY , class DATA >
  size_t **binarySize** (const std::map< KEY, DATA > &M)
- template<class T1 , class T2 >
  bool **writeBinary** (abstractCommHandle *fp, const std::pair< T1, T2 > &p)
- template<class T1 , class T2 >
  bool **readBinary** (abstractCommHandle *fp, std::pair< T1, T2 > &p)

- template<class T1 , class T2 >
  size_t **binarySize** (const std::pair< T1, T2 > &p)
- bool **writeBinary** (commUtil::abstractCommHandle *fp, const linalg::CSYS_KIND &e)
- bool **readBinary** (commUtil::abstractCommHandle *fp, linalg::CSYS_-KIND &e)
- void **write** (std::ostream &os, const linalg::CSYS_KIND &e) throw ()
- void **read** (std::istream &is, linalg::CSYS_KIND &e) throw ()
- std::ostream & **operator**<< (std::ostream &os, const linalg::CSYS_KIND &e)
- std::istream & **operator**>> (std::istream &is, linalg::CSYS_KIND &e)
- void **write** (std::ostream &os, const mere::Z &z)
- void **read** (std::istream &is, mere::Z &z)
- void **write** (std::ostream &os, const mere::R &r)
- void **read** (std::istream &is, mere::R &r)
- void **write** (std::ostream &os, const mere::C &c)
- void **read** (std::istream &is, mere::C &c)
- bool **writeBinary** (abstractCommHandle *fp, const mere::C &val)
- bool **readBinary** (abstractCommHandle *fp, mere::C &val)
- bool **writeBinary** (abstractCommHandle *fp, const mere::R &val)
- bool **readBinary** (abstractCommHandle *fp, mere::R &val)
- bool **writeBinary** (abstractCommHandle *fp, const mere::Z &val)
- bool **readBinary** (abstractCommHandle *fp, mere::Z &val)
- std::ostream & **operator**<< (std::ostream &os, const numerical_-functor::multipolePair &p)

- std::istream & **operator**>> (std::istream &is, numerical_-functor::multipolePair &p)
- std::ostream & **operator**<< (std::ostream &os, const numerical_-functor::multipoleSextuple &s)
- std::istream & **operator**>> (std::istream &is, numerical_-functor::multipoleSextuple &s)
- template<class T >
  bool **writeBinary** (abstractCommHandle ∗fp, const std::complex< T > &c)
- template<class T >
  bool **readBinary** (abstractCommHandle ∗fp, std::complex< T > &c)
- template<typename T >
  std::ostream & **operator**<< (std::ostream &os, const TMatrix::fieldPolarization< T > &beta)
- bool **writeBinary** (abstractCommHandle ∗fp, const TMatrix::VSW_KIND &eVSW)
- bool **readBinary** (abstractCommHandle ∗fp, TMatrix::VSW_KIND &eVSW)
- bool **writeBinary** (abstractCommHandle ∗fp, const TMatrix::REGULARITY_KIND &eReg)
- bool **readBinary** (abstractCommHandle ∗fp, TMatrix::REGULARITY_-KIND &eReg)
- template<typename T >
  std::ostream & **operator**<< (std::ostream &os, const TMatrix::vectorFieldKet< T > &ket)

- template<typename T >
  std::ostream & **operator**<< (std::ostream &os, const TMatrix::scalarFieldKet< T > &ket)

1.6.1.1 Detailed Description

Unified low-level binary interface to disk-resident, memory-resident, and MPI inter-process data transfer. Using these methods to implement “readBinary”, ”writeBinary”, and “binarySize” member or static methods for an end-user class is sufficient to provide unified disk, memory, and/or inter-process I/O of class instances.

1.6.2 distributed_solver_module Namespace Reference

Stand-alone interface to Trilinos Belos solvers: separates Trilinos classes from numerical_functor namespace.

Classes

- class distributedSolver_module

1.6.2.1 Detailed Description

Stand-alone interface to Trilinos Belos solvers: separates Trilinos classes from numerical_functor namespace. This namespace will provide functionality required for explicit instantiation of template < > distributedSolverFunctor<double> Modularization allows separation of incompletely templated Trilinos classes from the fully-templated classes in other namespaces.

1.6.3 geometry Namespace Reference

Region primitives and functions related to boundary-surface and lattice calculations.

Classes

- class icosphere
- class integerLattice
- class lattice
- class rectangularPrism
- class regionSelector
- class sphericalRegion
- class spheroidalRegion
- class cappedCylindricalRegion
- class combinedRegion
- class translatedRegion

Enumerations

- enum **spherical_coordinate** { **RADIUS**, **AZIMUTH**, **ZENITH** }
- enum **symmetry_kind** { **NO_SYMMETRY** = 0, **CONSTANT**, **SIGMA**, **ANTI_SIGMA** }

1.6.3.1 Detailed Description

Region primitives and functions related to boundary-surface and lattice calculations. Implementation note: associating the lattice with namespace geometry im-

plicitely associates lattice non-geometry related features, such as lattice::material_info and lattice::particle_info with this same namespace, which is potentially undesirable.

1.6.4 linalg Namespace Reference

Support for fully-templated, arbitrary precision linear algebra.

Namespaces

- namespace sparse

  Support for sparse and block-sparse vectors and matrices.

Classes

- class abstractMatrix
- class row_major_index
- class linear_interpolator

  N-dimensional linear interpolation of values of arbitrary rank.
  - *uses minimalist N-dimensional matrix semantics as (<linear container>, <shape ntuple>);*
  - *mesh corresponding to data instantiation is specified as ntuple_interval;*
  - *rank of value-type for N-dimensional matrix is arbitrary, but is a fixed-allocation type such as a scalar or ntuple<R,NDIM>.*

- class nonresidentObjectHandler
- class ntuple
- class ntuple_interval
- struct tensor_traits

Traits class to facilitate implementation of N-dimensional arrays and container objects with either scalar (RANK = 0) or non-scalar (RANK > 0) elements.

- struct tensor_traits< linalg::ntuple< T, NDIM > >

Enumerations

- enum **CSYS_KIND** { **NO_CSYS**, **CARTESIAN**, **CYLINDRICAL**, **SPHERICAL** }

Functions

- template<class N >

  std::vector< N > **primefactors** (N x)
- template<class N >

  void **primefactors** (N x, std::vector< N > &dest)
- template<class N >

  std::vector< N > **factors** (N x)
- template<class N >

  void **factors** (N x, std::vector< N > &dest)
- template<class N >

  std::vector< N > **uniform_factorization** (N x, size_t NDIM)
- template<class N >

  void **uniform_factorization** (N x, size_t NDIM, std::vector< N > &dest)
- template<class IT >

  std::vector< std::vector< typename std::iterator_traits< IT >::value_type > > **combinations** (IT itStart, IT itEnd, size_t n)

- template<class IT >

  void **combinations** (IT itStart, IT itEnd, size_t n, std::vector< std::vector< typename std::iterator_traits< IT >::value_type > > &dest)

- template<class IT >

  size_t **min_index** (IT itStart, IT itEnd)

- template<class IT >

  size_t **max_index** (IT itStart, IT itEnd)

- template<class U , class N >

  std::vector< N > argsort (const U &u)

  index sort:

- template<class U , class N >

  void **argsort** (const U &u, std::vector< N > &dest)

- template<class V >

  gmm::linalg_traits< V >::value_type product (const V &v, size_t beg_=0, size_t end_=static_cast< size_t >(-1))

  product of vector elements (i.e. $\Pi v_k : k \in [beg, end)$*):*

- size_t **row_major_index** (const std::vector< size_t > &indices, const std::vector< size_t > &shape)
- void **inverse_row_major_index** (size_t n, const std::vector< size_t > &shape, std::vector< size_t > &dest)
- std::vector< size_t > **inverse_row_major_index** (size_t n, const std::vector< size_t > &shape)
- bool **increment_row_major_indices** (const std::vector< size_t > &src, size_t ndim, const std::vector< std::pair< size_t, size_t > > &intervals, std::vector< size_t > &dest)

- template<class U >
  bool **any** (const U &u, bool(∗test_func)(const typename gmm::linalg_traits< U >::value_type &))
- template<class U >
  bool **all** (const U &u, bool(∗test_func)(const typename gmm::linalg_traits< U >::value_type &))
- template<class U >
  bool **any** (const U &u, int(∗test_func)(const typename gmm::linalg_traits< U >::value_type &))
- template<class U >
  bool **all** (const U &u, int(∗test_func)(const typename gmm::linalg_traits< U >::value_type &))
- template<class U >
  bool **any** (const U &u, bool(∗test_func)(const typename gmm::linalg_traits< U >::value_type &), gmm::abstract_vector)
- template<class U >
  bool **any** (const U &u, bool(∗test_func)(const typename gmm::linalg_traits< U >::value_type &), gmm::abstract_matrix)
- template<class U >
  bool **any** (const U &u, bool(∗test_func)(const typename gmm::linalg_traits< U >::value_type &), gmm::abstract_matrix, gmm::col_major)
- template<class U >
  bool **any** (const U &u, bool(∗test_func)(const typename gmm::linalg_traits< U >::value_type &), gmm::abstract_matrix, gmm::row_major)
- template<class U >
  bool **all** (const U &u, bool(∗test_func)(const typename gmm::linalg_traits<

U >::value_type &), gmm::abstract_vector)

- template<class U >

  bool **all** (const U &u, bool(∗test_func)(const typename gmm::linalg_traits< U >::value_type &), gmm::abstract_matrix)

- template<class U >

  bool **all** (const U &u, bool(∗test_func)(const typename gmm::linalg_traits< U >::value_type &), gmm::abstract_matrix, gmm::col_major)

- template<class U >

  bool **all** (const U &u, bool(∗test_func)(const typename gmm::linalg_traits< U >::value_type &), gmm::abstract_matrix, gmm::row_major)

- template<class U1 , class U2 >

  bool **sameOrigin** (const U1 &u1, const U2 &u2)

- template<class T >

  void **conv** (std::vector< T > &vT1, const std::vector< T > &vT2)

- template<class T >

  void **real** (std::vector< T > &vT1, const std::vector< T > &vT2)

- template<class T >

  void **imag** (std::vector< T > &vT1, const std::vector< T > &vT2)

- template<class T , class U >

  void **conv** (std::vector< T > &vT, const std::vector< U > &vU)

- template<class T , class U >

  void **real** (std::vector< T > &vT, const std::vector< U > &vU)

- template<class T , class U >

  void **imag** (std::vector< T > &vT, const std::vector< U > &vU)

- template<class V1 , class V2 , class V3 >

  void **elementwiseDiv** (const V1 &v1, const V2 &v2, V3 &v3) throw ()

- template<class V1 , class V2 , class V3 >

  void **elementwiseMult** (const V1 &v1, const V2 &v2, V3 &v3) throw ()

- template<class T1 , class T2 , class T3 >

  T3 **multiplies** (const T1 &, const T2 &)

- template<class T1 , class T2 , class T3 >

  T3 **divides** (const T1 &, const T2 &)

- template<class V1 , class V2 >

  void **diff** (const V1 &v1, V2 &v2, size_t nDiff=1)

- template<class T >

  void **transpose** (const gmm::dense_matrix< T > &src, gmm::dense_-matrix< T > &dest)

- template<class U >

  void **offset** (U &u, const typename gmm::linalg_traits< U >::value_type &delta)

- template<class U >

  void **offset** (U &u, const typename gmm::linalg_traits< U >::value_type &delta, gmm::abstract_vector)

- template<class U >

  void **offset** (U &u, const typename gmm::linalg_traits< U >::value_type &delta, gmm::abstract_matrix)

- template<class U >

  long **numberNonzeroElements** (const U &u, const typename gmm::number_traits< typename gmm::linalg_traits< U >::value_type >::magnitude_type &eps_=epsilon< typename gmm::number_traits< typename gmm::linalg_traits< U >::value_type >::magnitude_type >())

- template<class U >

  long **numberNonzeroElements** (const U &u, const typename gmm::number_traits< typename gmm::linalg_traits< U >::value_type >::magnitude_type &eps_, gmm::abstract_vector)

- template<class U >

  long **numberNonzeroElements** (const U &u, const typename gmm::number_traits< typename gmm::linalg_traits< U >::value_type >::magnitude_type &eps_, gmm::abstract_matrix)

- template<class U >

  void **fill** (U &u, const typename gmm::linalg_traits< U >::value_type &val)

- template<class U >

  void **fill** (U &u, const typename gmm::linalg_traits< U >::value_type &val, gmm::abstract_vector)

- template<class U >

  void **fill** (U &u, const typename gmm::linalg_traits< U >::value_type &val, gmm::abstract_matrix)

- template<class T , class U >

  void **iota** (typename U::iterator itStart, typename U::iterator itEnd, const T &initValue, const T &dt=one< T >())

- template<class U >

  void **initFromArray** (U &u, const typename gmm::linalg_traits< U >::value_type ∗val)

- template<class U >

  void **initFromArray** (U &u, const typename gmm::linalg_traits< U >::value_type ∗val, gmm::abstract_vector)

- template<class U >

  void **initFromArray** (U &u, const typename gmm::linalg_traits< U >::value_type *val, gmm::abstract_matrix)

- template<class V >

  void **e1** (V &v)

- template<class V >

  void **e2** (V &v)

- template<class V >

  void **e3** (V &v)

- template<class T , class U >

  void **conv** (gmm::dense_matrix< T > &aT, const gmm::dense_matrix< U > &aU)

- template<class T >

  void **saveVectorData** (const std::vector< T > &V, const std::string &sFileName) throw ()

- template<class T >

  void **loadVectorData** (std::vector< T > &V, const std::string &sFileName) throw ()

- template<class T1 , class T2 >

  void **conv** (std::complex< T1 > &t1, const T2 &t2)

- template<class T1 , class T2 >

  void **conv** (T1 &t1, const std::complex< T2 > &t2)

- template<class T1 , class T2 >

  void **conv** (std::complex< T1 > &t1, const std::complex< T2 > &t2)

- template<class T >

  const T **max_abs** (const T &a, const T &b)

- template<class T >

  const T **min_abs** (const T &a, const T &b)
- template<class T >

  const T **max_abs** (const T &a, const T &b, const T &c)
- template<class T >

  const T **min_abs** (const T &a, const T &b, const T &c)
- template<class T1 , class T2 >

  void **cart2sphere** (const T1 &x, const T1 &y, const T1 &z, T2 &r, T2 &theta, T2 &phi)
- template<class T1 , class T2 >

  void **sphere2cart** (const T1 &r, const T1 &theta, const T1 &phi, T2 &x, T2 &y, T2 &z)
- template<class T1 , class T2 >

  void **cart2sphere** (const std::vector< T1 > &vSrc, std::vector< T2 > &vDest)
- template<class T1 , class T2 >

  void **sphere2cart** (const std::vector< T1 > &vSrc, std::vector< T2 > &vDest)
- template<class T1 , class T2 >

  void **cart2sphere** (const ntuple< T1, 3 > &vSrc, ntuple< T2, 3 > &vDest)
- template<class T1 , class T2 >

  void **sphere2cart** (const ntuple< T1, 3 > &vSrc, ntuple< T2, 3 > &vDest)
- template<class T >

  void **cartVect2sphere** (const T &x, const T &y, const T &z, const T &vx, const T &vy, const T &vz, T &r, T &theta, T &phi, T &vr, T &vtheta, T &vphi)

- template<class T >

  void **sphereVect2cart** (const T &r, const T &theta, const T &phi, const T &vr, const T &vtheta, const T &vphi, T &x, T &y, T &z, T &vx, T &vy, T &vz)

- template<class T >

  void **cartVect2sphere** (const std::vector< T > &vSrcP, const std::vector< T > &vSrcV, std::vector< T > &vDestP, std::vector< T > &vDestV)

- template<class T >

  void **sphereVect2cart** (const std::vector< T > &vSrcP, const std::vector< T > &vSrcV, std::vector< T > &vDestP, std::vector< T > &vDestV)

- template<class V1 , class V2 >

  void quadraticFormula (const V1 &vP, V2 &vRoot, typename V1::value_-type &D)

  Solve the quadratic formula:

- template<class V1 >

  void quadraticDiscriminant (const V1 &vP, typename V1::value_type &D)

  Calculate the discriminant for the quadratic formula:

- template<class T >

  void **cubicFormula** (const std::vector< T > &vP, std::vector< std::complex< T > > &vRoot)

- template<class T >

  void **cubicFormula** (const std::vector< std::complex< T > > &vP, std::vector< std::complex< T > > &vRoot)

- template<class T >

  T **polyval** (const std::vector< T > &vP, const T &arg)

- template<class T >

  std::complex< T > **polyval** (const std::vector< std::complex< T > > &vP, const std::complex< T > &arg)

- template<class T >

  void sphericalToHelical (const T &v_r, const T &v_theta, const T &v_phi, const T &r, const T &theta, const T &phi, T &v_minus, T &v_0, T &v_plus, bool cosTheta=false) throw ()

  convert vector [v_r, v_theta, v_phi](r, theta, phi) to [v_minus, v_0, v_plus] in helical basis. optionally use u = cos(theta) as polar argument.

- template<class T >

  void sphericalToHelical (const T &v_r, const T &v_theta, const T &v_phi, const T &r, const T &theta, const long &m, T &v_minus, T &v_0, T &v_plus, bool cosTheta=false) throw ()

  convert vector [v_r, v_theta, v_phi](r, theta, phi) to [v_minus, v_0, v_plus] in helical basis. optionally use u = cos(theta) as polar argument; special usage: additional parsing by m-value.

- template<class R >

  R **randomCoord** (CSYS_KIND eCSYS, size_t dimOffset) throw ()

- template<class U , size_t NDIM>

  gmm::linalg_traits< U >::reference ND_deref (U &u, const ntuple< size_t, NDIM > &shape, const ntuple< size_t, NDIM > &n) throw (std::runtime_error)

  Dereference an N-dimensional matrix.

- template<class U , size_t NDIM>

  const gmm::linalg_traits< U >::reference ND_deref (const U &u, const

ntuple< size_t, NDIM > &shape, const ntuple< size_t, NDIM > &n) throw (std::runtime_error)

Dereference an N-dimensional matrix.

- template<class T >

  std::vector< T > central_diff_weights (size_t Np=3, size_t deriv_order=1) throw (std::runtime_error)

  Weights for central-difference numerical derivative calculation. param[in] Np number-of-points included in calculation of each derivative point param[in] deriv_order derivative order Np >= deriv_order+1, and must be odd.

- template<class U1 , class U2 , size_t NDIM>

  void central_diff (U1 &dest, const ntuple< size_t, NDIM > &shape, const U2 &src, size_t dim, size_t Np=3, size_t deriv_order=1, int BC_enum=0) throw (std::runtime_error)

  Central-difference numerical derivative calculation. param[in] src data points param[in] dest output points param[in] shape dimension limits for src and dest param[in] dim dimension over which to take derivative param[in] Np number-of-points included in calculation of each derivative point param[in] deriv_order derivative order param[in] boundary_condition enum: 0: periodic_BC

  - *Np >= deriv_order+1, and must be odd*
  - *At present, only implemented B.C. is periodic*

- template<class T , size_t DIM>

  ntuple< T, DIM > **operator**∗ (const T &r, const ntuple< T, DIM > &p)

- template<class T , size_t DIM>

  numberTraits< T >::magnitudeType **dist** (const ntuple< T, DIM > &p1, const ntuple< T, DIM > &p2)

- template<class T , size_t DIM>

  numberTraits< T >::magnitudeType **absSqr** (const ntuple< T, DIM > &p)

- template<class T , size_t DIM>

  size_t **numberNonzeroElements** (const ntuple< T, DIM > &p, const T &eps=epsilon< T >())
- template<class T , size_t DIM>

  bool **isZero** (const ntuple< T, DIM > &p, const T &eps=epsilon< T >())
- template<class T , size_t DIM>

  bool **isNAN** (const ntuple< T, DIM > &p)
- template<class T , size_t DIM>

  size_t **min_element** (const ntuple< T, DIM > &p)
- template<class T , size_t DIM>

  size_t **max_element** (const ntuple< T, DIM > &p)
- template<class T , size_t DIM>

  void **sort** (ntuple< T, DIM > &p) throw ()
- template<class T , size_t DIM>

  void **index_sort** (const ntuple< T, DIM > &p, ntuple< size_t, DIM > &vn) throw ()
- template<class T1 , class T2 , size_t DIM>

  void **conv** (ntuple< T1, DIM > &dest, const ntuple< T2, DIM > &src) throw ()
- template<class T , size_t DIM>

  bool **writeBinary** (abstractCommHandle *fp, const ntuple< T, DIM > &p)
- template<class T , size_t DIM>

  bool **readBinary** (abstractCommHandle *fp, ntuple< T, DIM > &p)
- template<class T , size_t DIM>

  std::ostream & **operator**<< (std::ostream &os, const ntuple< T, DIM > &p)

- template<class T , size_t DIM>

  std::istream & **operator**>> (std::istream &is, ntuple< T, DIM > &p)

- template<class T , size_t DIM>

  bool **writeBinary** (abstractCommHandle *fp, const ntuple_interval< T, DIM > &p)

- template<class T , size_t DIM>

  bool **readBinary** (abstractCommHandle *fp, ntuple_interval< T, DIM > &p)

- template<class T , size_t DIM>

  std::ostream & **operator**<< (std::ostream &os, const ntuple_interval< T, DIM > &p)

1.6.4.1 Detailed Description

Support for fully-templated, arbitrary precision linear algebra.

1.6.4.2 Function Documentation

1.6.4.3 template<class U , size_t NDIM> gmm::linalg_traits<U>::reference linalg::ND_deref (U & *u*, const ntuple< size_t, NDIM > & *shape*, const ntuple< size_t, NDIM > & *n*) throw (std::runtime_error)

Dereference an N-dimensional matrix.

Template Parameters

U a linear container class with fixed-allocation value-type (e.g. scalar, ntuple, or tensor)

NDIM number of dimensions of this matrix

Parameters

[in] ***u*** container

[in] ***shape*** dimension limits

[in] ***n*** ntuple index of element to reference

1.6.4.4 template<class U , size_t NDIM> const gmm::linalg_traits<U>::reference linalg::ND_deref (const U & *u*, const ntuple< size_t, NDIM > & *shape*, const ntuple< size_t, NDIM > & *n*) throw (std::runtime_error)

Dereference an N-dimensional matrix.

Template Parameters

U a linear container class with fixed-allocation value-type (e.g. scalar, ntuple, or tensor)

NDIM number of dimensions of this matrix

Parameters

[in] ***u*** container

[in] ***shape*** dimension limits

[in] ***n*** ntuple index of element to reference

1.6.4.5 template<class V1 > void linalg::quadraticDiscriminant (const V1 & *vP*, typename V1::value_type & *D*)

Calculate the discriminant for the quadratic formula:

Template Parameters

V1 container type of value_type real or complex

Parameters

[in] ***vP*** vector of polynomial coefficients

[out] ***D*** discriminant

1.6.4.6 template<class V1 , class V2 > void linalg::quadraticFormula (const V1 & *vP*, V2 & *vRoot*, typename V1::value_type & *D*)

Solve the quadratic formula:

Template Parameters

V1 container type of value_type real or complex

V2 container type of value_type complex (or value_type real $\Leftrightarrow$ roots are somehow constrained to be real)

Parameters

[in] ***vP*** vector of polynomial coefficients

[out] ***vRoot*** vector of roots

[out] ***D*** discriminant

1.6.5 linalg::sparse Namespace Reference

Support for sparse and block-sparse vectors and matrices.

Classes

- class index2DFunctor
- class rowMajorIndex2DFunctor
- class compressed2DIndex
- class blockMatrix
- class compressed1DIndex
- class blockMatrixHelper
- class blockVector
- class nDimensionalKey
- class sparseNDimensional_iterator
- struct maskOtherDim
- struct lessThanDim1
- struct lessThanDim2
- class indefiniteVector
- class indefiniteSparseMatrix
- class sparseSubdimensionReference

Functions

- template<class T >
  void **mult** (const blockMatrix< T > &src1, const blockMatrix< T > &src2, blockMatrix< T > &dest)

- template<class T >
  void **mult** (const blockMatrix< T > &src1, const blockVector< T > &src2, blockVector< T > &dest)
- template<class T , class U >
  void **conv** (blockMatrix< T > &aT, const blockMatrix< U > &aU)
- template<class T , class U >
  void **conv** (blockVector< T > &aT, const blockVector< U > &aU)

1.6.5.1 Detailed Description

Support for sparse and block-sparse vectors and matrices.

1.6.6 mere Namespace Reference

Unified, minimalist, and thread-safe, arbitrary precision integer, real, and complex number classes.

Classes

- class mere_exception
- class mere_base
- class Z
- class R
- class C

Functions

- Z **mod** (const Z &num, const Z &den)
- Z **rem** (const Z &num, const Z &den)

- bool **isOdd** (const Z &z)
- bool **isEven** (const Z &z)
- Z **abs** (const Z &z)
- Z **neg** (const Z &z)
- Z **power** (const Z &z, unsigned long e)
- Z **power2** (const Z &z, long e)
- std::istream & **operator**>> (std::istream &is, Z &z)
- std::ostream & **operator**<< (std::ostream &os, const Z &z)
- void **conv** (Z &z, long n)
- void **conv** (Z &z, unsigned long u)
- void **conv** (long &n, const Z &z)
- void **conv** (unsigned long &u, const Z &z)
- void **swap** (Z &z1, Z &z2)
- bool **operator==** (long n, const Z &z)
- bool **operator!=** (long n, const Z &z)
- bool **operator**<= (long n, const Z &z)
- bool **operator**>= (long n, const Z &z)
- bool **operator**< (long n, const Z &z)
- bool **operator**> (long n, const Z &z)
- Z **operator+** (long n, const Z &z)
- Z **operator-** (long n, const Z &z)
- Z **operator**∗ (long n, const Z &z)
- Z **operator/** (long n, const Z &z)
- Z **operator%** (long n, const Z &z)

- R **operator+** (const R &r, const Z &z)
- R **operator+** (const Z &z, const R &r)
- R **operator-** (const R &r, const Z &z)
- R **operator-** (const Z &z, const R &r)
- R **operator**∗ (const R &r, const Z &z)
- R **operator**∗ (const Z &z, const R &r)
- R **operator/** (const R &r, const Z &z)
- R **operator/** (const Z &z, const R &r)
- R **mod** (const R &num, const R &den)
- R **rem** (const R &num, const R &den)
- R **abs** (const R &r)
- R **neg** (const R &r)
- R **inv** (const R &r)
- R **sqr** (const R &r)
- R **sqrt** (const R &r)
- R **cbrt** (const R &r)
- R **root** (const R &r, unsigned long k)
- R **pow** (const R &r, const R &e)
- R **power** (const R &r, long e)
- R **power** (const R &r, const Z &e)
- R **log** (const R &r)
- R **log2** (const R &r)
- R **log10** (const R &r)
- R **exp** (const R &r)

- R **exp2** (const R &r)
- R **exp10** (const R &r)
- R **cos** (const R &r)
- R **sin** (const R &r)
- R **tan** (const R &r)
- R **sec** (const R &r)
- R **csc** (const R &r)
- R **cot** (const R &r)
- R **acos** (const R &r)
- R **asin** (const R &r)
- R **atan** (const R &r)
- R **gamma** (const R &r)
- R **tgamma** (const R &r)
- R **atan2** (const R &y, const R &x)
- R **hypot** (const R &y, const R &x)
- void **mod2Pi** (R &r)
- R **cosh** (const R &r)
- R **sinh** (const R &r)
- R **tanh** (const R &r)
- R **sech** (const R &r)
- R **csch** (const R &r)
- R **coth** (const R &r)
- R **acosh** (const R &r)
- R **asinh** (const R &r)

- R **atanh** (const R &r)
- R **log2** (void)
- R **log10** (void)
- R **pi** (void)
- R **euler** (void)
- R **catalan** (void)
- R **ceil** (const R &r)
- R **floor** (const R &r)
- R **round** (const R &r)
- R **trunc** (const R &r)
- std::istream & **operator**>> (std::istream &is, R &r)
- std::ostream & **operator**<< (std::ostream &os, const R &r)
- void **conv** (R &r, const Z &z)
- void **conv** (R &r, long n)
- void **conv** (R &r, unsigned long u)
- void **conv** (R &r, double d)
- void **conv** (double &d, const R &r)
- void **conv** (long &n, const R &r)
- void **conv** (Z &z, const R &r)
- void **swap** (R &r1, R &r2)
- bool **operator==** (const Z &z, const R &r)
- bool **operator!=** (const Z &z, const R &r)
- bool **operator**<= (const Z &z, const R &r)
- bool **operator**>= (const Z &z, const R &r)

- bool **operator**< (const Z &z, const R &r)
- bool **operator**> (const Z &z, const R &r)
- bool **operator==** (long n, const R &r)
- bool **operator!=** (long n, const R &r)
- bool **operator**<= (long n, const R &r)
- bool **operator**>= (long n, const R &r)
- bool **operator**< (long n, const R &r)
- bool **operator**> (long n, const R &r)
- R **operator+** (long n, const R &r)
- R **operator-** (long n, const R &r)
- R **operator**∗ (long n, const R &r)
- R **operator/** (long n, const R &r)
- C **operator+** (const C &c, const Z &z)
- C **operator+** (const Z &z, const C &c)
- C **operator-** (const C &c, const Z &z)
- C **operator-** (const Z &z, const C &c)
- C **operator**∗ (const C &c, const Z &z)
- C **operator**∗ (const Z &z, const C &c)
- C **operator/** (const C &c, const Z &z)
- C **operator/** (const Z &z, const C &c)
- C **operator+** (const C &c, const R &r)
- C **operator+** (const R &r, const C &c)
- C **operator-** (const C &c, const R &r)
- C **operator-** (const R &r, const C &c)

- C **operator**∗ (const C &c, const R &r)
- C **operator**∗ (const R &r, const C &c)
- C **operator/** (const C &c, const R &r)
- C **operator/** (const R &r, const C &c)
- R **abs** (const C &c)
- R **arg** (const C &c)
- C **neg** (const C &c)
- C **inv** (const C &c)
- C **conj** (const C &c)
- C **polar** (const R &magnitude, const R &argument)
- C **sqr** (const C &c)
- C **sqrt** (const C &c)
- C **cbrt** (const C &c)
- C **root** (const C &c, unsigned long k)
- C **pow** (const C &c, const C &e)
- C **power** (const C &c, long e)
- C **power** (const C &c, const Z &e)
- C **log** (const C &c)
- C **log2** (const C &c)
- C **log10** (const C &c)
- C **exp** (const C &c)
- C **exp2** (const C &c)
- C **exp10** (const C &c)
- C **cos** (const C &c)

- C **sin** (const C &c)
- C **tan** (const C &c)
- C **sec** (const C &c)
- C **csc** (const C &c)
- C **cot** (const C &c)
- C **acos** (const C &c)
- C **asin** (const C &c)
- C **atan** (const C &c)
- C **gamma** (const C &c)
- C **tgamma** (const C &c)
- C **atan2** (const C &y, const C &x)
- C **cosh** (const C &c)
- C **sinh** (const C &c)
- C **tanh** (const C &c)
- C **sech** (const C &c)
- C **csch** (const C &c)
- C **coth** (const C &c)
- C **acosh** (const C &c)
- C **asinh** (const C &c)
- C **atanh** (const C &c)
- std::istream & **operator**>> (std::istream &is, C &c)
- std::ostream & **operator**<< (std::ostream &os, const C &c)
- void **conv** (C &c, const Z &z)
- void **conv** (C &c, long n)

- void **conv** (C &c, unsigned long u)
- void **conv** (C &c, const R &r)
- void **conv** (C &c, double d)
- void **swap** (C &c1, C &c2)
- C **operator+** (long n, const C &c)
- C **operator-** (long n, const C &c)
- C **operator*** (long n, const C &c)
- C **operator/** (long n, const C &c)

1.6.6.1 Detailed Description

Unified, minimalist, and thread-safe, arbitrary precision integer, real, and complex number classes.

1.6.7 number Namespace Reference

Interface traits and primitive functions for POD and arbitrary-precision number classes.

Classes

- class numberTraits< mere::Z >
- class numberTraits< mere::R >
- class numberTraits< mere::C >
- class numberTraits

  traits class enforcing number system heirarchy.

Functions

- std::ostream & **abortMsgStream** (bool bAppend=false)
- const mere::R & **real** (const mere::C &c)
- const mere::R & **imag** (const mere::C &c)
- const mere::R & **real** (const mere::R &c)
- const mere::R & **imag** (const mere::R &c)
- mere::R **conj** (const mere::R &r)
- mere::C **sign** (const mere::C &c)
- mere::R **sign** (const mere::R &r)
- mere::Z **sign** (const mere::Z &z)
- template<>
  numberTraits< mere::Z >::magnitudeType **sqrNorm**< **mere::Z** > (const mere::Z &z)
- template<>
  numberTraits< mere::R >::magnitudeType **sqrNorm**< **mere::R** > (const mere::R &r)
- template<>
  numberTraits< mere::C >::magnitudeType **sqrNorm**< **mere::C** > (const mere::C &c)
- template<>
  mere::C **inv**< **mere::C** > (const mere::C &c)
- template<>
  mere::R **inv**< **mere::R** > (const mere::R &r)
- mere::R **fabs** (const mere::C &c)
- mere::R **fabs** (const mere::R &r)

- long **integer_digits** (const mere::Z &Z)
- long **bitsToDigits** (const long &prec_)
- template<>
  long **precision**< **mere::C** > (const mere::C &c)
- template<>
  long **precision**< **mere::C** > (void)
- template<>
  long **precision**< **mere::R** > (const mere::R &r)
- template<>
  long **precision**< **mere::R** > (void)
- template<>
  long **precision**< **mere::Z** > (void)
- template<>
  long **precision**< **mere::Z** > (const mere::Z &r)
- template<>
  numberTraits< numberTraits< mere::C >::magnitudeType >::POD_-
  VALUE_RETURN **epsilon**< **mere::C** > (void)
- template<>
  numberTraits< numberTraits< mere::R >::magnitudeType >::POD_-
  VALUE_RETURN **epsilon**< **mere::R** > (void)
- template<>
  numberTraits< numberTraits< mere::Z >::magnitudeType >::POD_-
  VALUE_RETURN **epsilon**< **mere::Z** > (void)
- template<>
  numberTraits< mere::C >::magnitudeType **root2Epsilon**< **mere::C** >
  (void)

- template<>
  numberTraits< mere::R >::magnitudeType **root2Epsilon**< **mere::R** >
  (void)
- template<>
  numberTraits< mere::C >::magnitudeType **root4Epsilon**< **mere::C** >
  (void)
- template<>
  numberTraits< mere::R >::magnitudeType **root4Epsilon**< **mere::R** >
  (void)
- template<>
  numberTraits< mere::C >::POD_VALUE_RETURN **one**< **mere::C** >
  (void)
- template<>
  numberTraits< mere::C >::POD_VALUE_RETURN **one_i**< **mere::C** >
  (void)
- template<>
  numberTraits< mere::R >::POD_VALUE_RETURN **one**< **mere::R** >
  (void)
- template<>
  numberTraits< mere::C >::POD_VALUE_RETURN **zero**< **mere::C** >
  (void)
- template<>
  numberTraits< mere::R >::POD_VALUE_RETURN **zero**< **mere::R** >
  (void)
- template<>
  numberTraits< numberTraits< mere::C >::magnitudeType >::POD_-

VALUE_RETURN **pi**< **mere::C** > (void)

- template<>
  numberTraits< numberTraits< mere::R >::magnitudeType >::POD_-
  VALUE_RETURN **pi**< **mere::R** > (void)
- template<>
  numberTraits< mere::Z >::POD_VALUE_RETURN **one**< **mere::Z** >
  (void)
- template<>
  numberTraits< mere::Z >::POD_VALUE_RETURN **zero**< **mere::Z** >
  (void)
- template<>
  mere::C **random**< **mere::C** > (void)
- template<>
  void **seedRandom**< **mere::C** > (void) throw ()
- template<>
  void **seedRandom**< **mere::C** > (const std::string &seed) throw ()
- template<>
  mere::R **random**< **mere::R** > (void)
- template<>
  void **seedRandom**< **mere::R** > (void) throw ()
- template<>
  void **seedRandom**< **mere::R** > (const std::string &seed) throw ()
- template<>
  mere::R **normalDist**< **mere::R** > (const mere::R &mean, const mere::R
  &variance) throw ()

- template<>

  mere::Z **random**< **mere::Z** > (void)
- template<>

  void **seedRandom**< **mere::Z** > (void) throw ()
- template<>

  void **seedRandom**< **mere::Z** > (const std::string &seed) throw ()
- template<>

  void **conv**< **mere::C, unsigned long** > (mere::C &t1, const unsigned long &t2)
- template<>

  void **conv**< **mere::C, mere::Z** > (mere::C &t1, const mere::Z &t2)
- template<>

  void **conv**< **mere::C, long** > (mere::C &t1, const long &t2)
- template<>

  void **conv**< **mere::C, int** > (mere::C &t1, const int &t2)
- template<>

  void **conv**< **mere::C, double** > (mere::C &t1, const double &t2)
- template<>

  void **conv**< **mere::R, unsigned long** > (mere::R &t1, const unsigned long &t2)
- template<>

  void **conv**< **mere::R, mere::Z** > (mere::R &t1, const mere::Z &t2)
- template<>

  void **conv**< **mere::R, long** > (mere::R &t1, const long &t2)
- template<>

  void **conv**< **mere::R, int** > (mere::R &t1, const int &t2)

- template<>

  void **conv**< **mere::R, double** > (mere::R &t1, const double &t2)
- template<>

  void **conv**< **double, mere::R** > (double &t1, const mere::R &t2)
- template<>

  void **conv**< **mere::Z, unsigned long** > (mere::Z &t1, const unsigned long &t2)
- template<>

  void **conv**< **mere::Z, long** > (mere::Z &t1, const long &t2)
- template<>

  void **conv**< **mere::Z, int** > (mere::Z &t1, const int &t2)
- template<>

  void **conv**< **double, mere::Z** > (double &t1, const mere::Z &t2)
- template<>

  void **conv**< **long, mere::Z** > (long &t1, const mere::Z &t2)
- template<>

  void **conv**< **mere::Z, mere::R** > (mere::Z &t1, const mere::R &t2)
- template<>

  void **conv**< **long, mere::R** > (long &t1, const mere::R &t2)
- template<>

  void **setDefaultPrecision**< **mere::C** > (const long &prec)
- template<>

  long **getDefaultPrecision**< **mere::C** > (void)
- template<>

  void **setDefaultPrecision**< **mere::R** > (const long &prec)

- template<>

  long **getDefaultPrecision**< **mere::R** > (void)

- template<class T >

  void setDefaultPrecision (const long &prec)

- template<class T >

  long **getDefaultPrecision** (void)

- template<class T >

  long **precision** (const T &t)

- template<class T >

  long **precision** (void)

- template<class T >

  numberTraits< typename numberTraits< T >::magnitudeType >::POD_-VALUE_RETURN **epsilon** (void)

- template<class T >

  numberTraits< T >::magnitudeType **root2Epsilon** (void)

- template<class T >

  numberTraits< T >::magnitudeType **root4Epsilon** (void)

- template<class T >

  numberTraits< T >::POD_VALUE_RETURN **one** (void)

- template<class T >

  numberTraits< T >::POD_VALUE_RETURN **one_i** (void)

- template<class T >

  numberTraits< T >::POD_VALUE_RETURN **zero** (void)

- template<class T >

  numberTraits< typename numberTraits< T >::magnitudeType >::POD_-VALUE_RETURN **pi** (void)

- template<class T >

  T **random** (void)

- template<class T >

  void **seedRandom** (void) throw ()

- template<class T >

  void **seedRandom** (const std::string &seed) throw ()

- template<class R >

  R **normalDist** (const R &mean, const R &variance) throw ()

- template<class T1 , class T2 >

  void **conv** (T1 &t1, const T2 &t2)

- template<class T >

  T **sqr** (const T &arg)

- template<class T >

  T **cube** (const T &arg)

- template<class T >

  numberTraits< T >::magnitudeType **sqrNorm** (const T &arg)

- template<class T >

  T **minusOnePow_n** (const long &arg)

- template<class T >

  T **inv** (const T &t)

- template<class T >

  T **pow_n** (const T &arg, const long power)

- template<class T >

  bool **lesserMagnitude** (const T &a, const T &b)

- template<class T >

  bool **greaterMagnitude** (const T &a, const T &b)

- template<class T >

  T **integer** (const long &arg)

- template<class T , class U >

  T **ratio** (const U &num, const U &denom)

- template<class T >

  T **prod** (const long &min, const long &max)

- template<class T >

  numberTraits< T >::magnitudeType **norm** (const T &t)

- template<class T >

  numberTraits< T >::magnitudeType **norm** (const T &x, const T &y, const T &z)

- template<class T >

  const T **max** (const T &a, const T &b)

- template<class T >

  const T **min** (const T &a, const T &b)

- template<class T >

  const T **max** (const T &a, const T &b, const T &c)

- template<class T >

  const T **min** (const T &a, const T &b, const T &c)

- template<class T >

  bool **inDomain** (const T &val, const T &min, const T &max, bool interior=false)

- template<class T1 , class T2 >

  numberTraits< T1 >::magnitudeType **hypot** (const T1 &t1, const T2 &t2)

- template<class T >

  bool **isnan** (const T &t)

- template<>

  bool **isnan**< **double** > (const double &t)

- template<>

  bool **isnan**< **std::complex**< **double** > > (const std::complex< double > &t)

- template<class T >

  T **negativeNonZero** (const T &t)

1.6.7.1 Detailed Description

Interface traits and primitive functions for POD and arbitrary-precision number classes.

1.6.7.2 Function Documentation

1.6.7.3 template<class T > void number::setDefaultPrecision (const long & *prec*)

1.6.8 numerical_functor Namespace Reference

Classes and methods supporting object-oriented calculations of mathematical special functions.

Classes

- class additionCoefficientFunctor

- class azimuthalIntegrandFunctor
- class sbesselFunctor
- class cubicInterpolationFunctor
- class distributedSolverFunctor
- class distributedSolverFunctor< std::complex< double > >
- class eulerRotationFunctor
- class factorialFunctor
- class regionKet
- class fieldVisualizer
- class gauntFunctor
- class jacobiFunctor
- class legendreFunctor
- class multipolePair
- class multipoleSextuple
- class numericalFunctor_parameters_base
- class numericalFunctor
- class numericalFunctorLookupTable
- class timestamp
- class permittivityFunctor
- class continuumFunctor
- class discreteFunctor
- class quadratureFunctor
- class gaussLobattoQuadratureFunctor
- class tanh_sinh_quadratureFunctor

- class rotationCoefficientFunctor
- class scalarNewtonsFunctor
- class stirlingsAppxFunctor
- class vectorSphericalHarmonicFunctor
- class wigner3JFunctor

1.6.8.1 Detailed Description

Classes and methods supporting object-oriented calculations of mathematical special functions.

1.6.9 parallelUtil Namespace Reference

Support classes and methods for object-oriented multi-thread and multi-process coding.

Classes

- class dispatcher

  dispatcher/receiver task-based MPI implementation.

- class monolithic_dispatcher

  Single-task dispatcher/receiver MPI implementation.

- class parallelFunctor

  generic functor to MPI-task mapping, using no shared workspace.

- class loopIteratorList
- class loopIteratorArray

- class multiMutex

Functions

- template<class WS , class IT1 , class IT2 , class U1 , class U2 >
  bool **OMP_parallel_for** (void(∗threadFunc)(WS &, const loopIteratorList< IT1, IT2, U1, U2 > &) throw(std::string), WS &workspace, const loopIteratorList< IT1, IT2, U1, U2 > &indices)
- template<class WS , class IT1 , class IT2 , class U1 , class U2 >
  void ∗ **OMP_parallel_for_aux_const** (void ∗pvArgs)
- template<class WS , class IT1 , class IT2 , class U1 , class U2 >
  bool **OMP_parallel_for** (void(∗threadFunc)(WS &, loopIteratorList< IT1, IT2, U1, U2 > &) throw(std::string), WS &workspace, loopIteratorList< IT1, IT2, U1, U2 > &indices)
- template<class WS , class IT1 , class IT2 , class U1 , class U2 >
  void ∗ **OMP_parallel_for_aux_non_const** (void ∗pvArgs)
- size_t **OMP_NUM_THREADS** (void)
- template<class CLASS_ , class RVAL , class MEMBER_FUNC , class IT >
  std::vector< RVAL > **parallel_for** (CLASS_ ∗instance, MEMBER_FUNC threadFunc, IT begin_, IT end_, size_t N_THREAD=OMP_NUM_THREADS()) throw (std::string)
- template<class CLASS_ , class RVAL , class MEMBER_FUNC , class IT >
  void ∗ **parallel_for_aux** (void ∗pvArgs)
- template<class CLASS_ , class RVAL , class MEMBER_FUNC , class IT , class ARG >
  std::vector< RVAL > **parallel_for** (CLASS_ ∗instance, MEMBER_FUNC threadFunc, IT begin_, IT end_, ARG arg, size_t N_THREAD=OMP_NUM_THREADS()) throw (std::string)

- template<class CLASS_ , class RVAL , class MEMBER_FUNC , class IT , class ARG > void * **parallel_for_aux** (void *pvArgs)
- bool **is_process_root** (void)

1.6.9.1 Detailed Description

Support classes and methods for object-oriented multi-thread and multi-process coding. Emphasis is on providing a minimal set of efficient methods, with an interface as unified as possible. Methods are provided to supercede standard OMP pragmas, and to assist in making MPI utilization completely transparent.

1.6.10 TMatrix Namespace Reference

Classes and utility methods specific to single and multiple-particle T-matrix calculation.

Namespaces

- namespace constants

  Experimental data and constants.

Classes

- class multiCentered_tmatrix
- class clusterCentered_tmatrix
- class aggregate_cc_functor
- class aggregate_mc_functor
- class factoredPropagator

- class laguerreGaussianModeKet
- class planeWaveKet
- class mie_tmatrix
- struct position_ket
- struct multiCentered_vectorFieldKet
- class propagator_tmatrix
- class rotator_tmatrix
- class fieldPolarization
- class tmatrixIndex
- class fieldKetBase
- class vectorFieldKet
- class multiCentered_fieldKet
- class scalarFieldKet
- class tmatrix

Enumerations

- enum **REGULARITY_KIND** {

  REGULAR = 0, **OUTGOING** = 1, **INCOMING** = 2, **N_REGULARITY_-KIND** = 3,

  REGULAR = 0, **OUTGOING** = 1, **INCOMING** = 2, **N_REGULARITY_-KIND** = 3 }
- enum **VSW_KIND** {

  VSW_M = 0, **VSW_N** = 1, **VSW_L** = 2, **N_TRANSVERSE_VSW_KIND** = 2,

N_VSW_KIND = N_TRANSVERSE_VSW_KIND, **NO_VSW_KIND** = 3 }

- enum **REGULARITY_KIND** {

  REGULAR = 0, **OUTGOING** = 1, **INCOMING** = 2, **N_REGULARITY_-KIND** = 3,

  REGULAR = 0, **OUTGOING** = 1, **INCOMING** = 2, **N_REGULARITY_-KIND** = 3 }

Functions

- REGULARITY_KIND & **operator++** (REGULARITY_KIND &eReg)
- REGULARITY_KIND **operator++** (REGULARITY_KIND &eReg, int)
- REGULARITY_KIND & **operator--** (REGULARITY_KIND &eReg)
- REGULARITY_KIND **operator--** (REGULARITY_KIND &eReg, int)
- VSW_KIND & **operator++** (VSW_KIND &eVSW)
- VSW_KIND **operator++** (VSW_KIND &eVSW, int)
- VSW_KIND & **operator--** (VSW_KIND &eVSW)
- VSW_KIND **operator--** (VSW_KIND &eVSW, int)

1.6.10.1 Detailed Description

Classes and utility methods specific to single and multiple-particle T-matrix calculation. Wherever possible, codes with more general use, such as matrix utility methods, are implemented to be independent from, or at least to have restricted dependence on this namespace.

1.6.11 TMatrix::constants Namespace Reference

Experimental data and constants.

Classes

- class permittivityData

Variables

- const double **e** = 1.60217653e-19
- const double **hbar** = 1.05457168e-34 / e
- const double **c** = 299792458.0
- const double **epsilon_0** = 8.854187817e-12
- const double **m_e** = 9.1093826e-31
- const double **N_A** = 6.0221415e23

1.6.11.1 Detailed Description

Experimental data and constants. Instantiation of vectors of complex experimental dielectric permittivity data, along with selected material constants generally required for use. Materials included are Cu, Ag, Au, with abscissa in eV, and H_20 with abscissa in micron wavelength.

1.7 Class Documentation

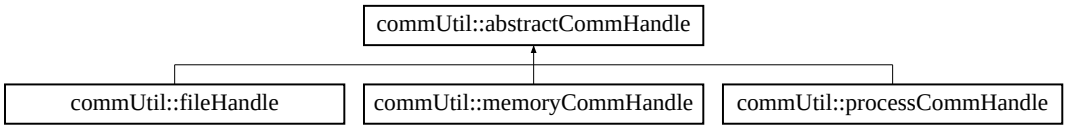
1.7.1 hash_map Class Reference

Inherited by linalg::sparse::indefiniteSparseMatrix< typename continuumFunctor< domainType, rangeType >::quadratureCacheNode * >, and linalg::sparse::indefiniteSparseMatrix< typename continuumFunctor< T, T >::quadratureCacheNode * >.

1.7.2 commUtil::abstractCommHandle Class Reference

An abstract base class to unify functionality of binary-I/O streams. These classes include: file-streams, MPI::communicator-streams, and memory I/O-streams.

Inheritance diagram for commUtil::abstractCommHandle:



Classes

- class **buffer**

Public Member Functions

- virtual size_t **read** (void *ptr, size_t size, size_t nitems)=0 throw ()

- virtual size_t **write** (const void ∗ptr, size_t size, size_t nitems)=0 throw ()
- virtual void **close** (void) throw ()
- void **sync** (void) throw ()
- void **flush** (void) throw ()
- bool **buffered** (void) const
- bool **buffered** (bool flag, size_t bytes=10 ∗1024 ∗1024) throw ()
- bool **buffered** (unsigned char ∗begin, unsigned char ∗end) throw ()
- bool **buffered** (unsigned char ∗beginRead, unsigned char ∗endRead, unsigned char ∗beginWrite, unsigned char ∗endWrite) throw ()

Static Public Member Functions

- static void **close** (abstractCommHandle ∗&h) throw ()

Protected Types

- enum **COMM_FLAGS** { **ANY_SET** = 0x0003, **READING** = 0x0001, **WRITING** = 0x0002 }

Protected Member Functions

- bool **valid** (void) const
- bool **valid** (bool state)
- unsigned short **flags** (void) const
- unsigned short **flags** (unsigned short newFlags)
- bool **reading** (void) const
- bool **writing** (void) const
- size_t **bufferedRead** (void ∗ptr, size_t size, size_t nitems) throw ()

- size_t **bufferedWrite** (const void *ptr, size_t size, size_t nitems) throw ()
- bool **readBuffered** (bool ignore_sync=false) const
- bool **writeBuffered** (bool ignore_flush=false) const
- size_t **readBufAvail** (void) const
- size_t **writeBufAvail** (void) const
- virtual bool **transferBufferHeader** (void) const
- virtual bool **partialBufferTransfer** (void) const

1.7.2.1 Detailed Description

An abstract base class to unify functionality of binary-I/O streams. These classes include: file-streams, MPI::communicator-streams, and memory I/O-streams.

1.7.3 commUtil::abstractCommHandle::buffer::header Struct Reference

Public Attributes

- size_t **size_**

1.7.4 commUtil::comm_error Class Reference

Base-class for binary-I/O exceptions.

Public Types

- typedef std::runtime_error **base_class**

Public Member Functions

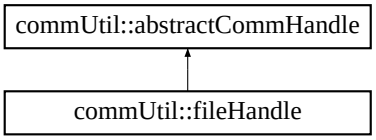
- comm_error & **operator=** (const comm_error &other) throw ()
- **comm_error** (const comm_error &other) throw ()
- **comm_error** (const std::string &msg) throw ()

1.7.4.1 Detailed Description

Base-class for binary-I/O exceptions.

1.7.5 commUtil::fileHandle Class Reference

Inheritance diagram for commUtil::fileHandle:



Public Member Functions

- int **fd** (void) const
- virtual size_t **read** (void ∗ptr, size_t size, size_t nitems) throw ()
- virtual size_t **write** (const void ∗ptr, size_t size, size_t nitems) throw ()
- virtual void **close** (void) throw ()
- void **seek** (long offset, int whence) throw ()
- long **tell** (void) throw ()

Static Public Member Functions

- static fileHandle ∗ **open** (const char ∗pathname, int flags, mode_t mode) throw ()

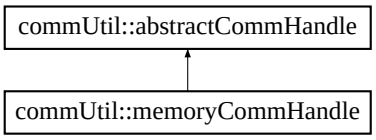
- static fileHandle * **open** (const char *filename, const char *mode) throw ()
- static bool **exists** (const char *fileName) throw ()
- static bool **isDirectory** (const char *fileName) throw ()
- static size_t **sizeOnMedia** (const char *fileName) throw ()
- static fileHandle * **clone** (FILE *fp) throw ()
- static fileHandle * **clone** (int fd) throw ()

Protected Member Functions

- virtual bool **transferBufferHeader** (void) const
- virtual bool **partialBufferTransfer** (void) const

1.7.6 commUtil::memoryCommHandle Class Reference

Inheritance diagram for commUtil::memoryCommHandle:



Public Member Functions

- virtual size_t **read** (void *ptr, size_t size, size_t nitems) throw ()
- virtual size_t **write** (const void *ptr, size_t size, size_t nitems) throw ()

Static Public Member Functions

- static memoryCommHandle ∗ **open** (void ∗buffer, size_t buf_size, const char ∗mode) throw ()

Protected Member Functions

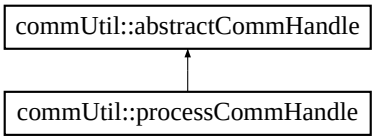
- virtual bool **transferBufferHeader** (void) const
- virtual bool **partialBufferTransfer** (void) const
- size_t **buf_remaining** (void) const

1.7.6.1 Detailed Description

The following class "memoryCommHandle" implements a uniform interface to abstractCommHandle to allow reading and writing to in-memory buffers. The primary purpose of this class is to allow binary format conversion for complicated objects to occur entirely in user or kernel space, at the time of _submission_ of an asynchronous-IO request, without the previous requirement for user-kernel transitions occuring at the moment of the actual I/O action (i.e. when this conversion occured previously in the AIO signal handler).

1.7.7 commUtil::processCommHandle Class Reference

Inheritance diagram for commUtil::processCommHandle:



Public Types

- enum **TRANSFER_KIND** { **POINT2POINT** = 0, **BROADCAST** = 1 }

Public Member Functions

- int **rank** (void) const
- const MPI::Comm & **comm** (void) const
- int **tag** (void) const
- TRANSFER_KIND **transfer_type** (void) const
- TRANSFER_KIND **transfer_type** (TRANSFER_KIND newTransferType)

- int **tag** (int newTag)
- virtual size_t **read** (void ∗ptr, size_t size, size_t nitems) throw ()
- virtual size_t **write** (const void ∗ptr, size_t size, size_t nitems) throw ()
- template<class U >
  void **gather** (const U &u) throw ()
- template<class U >
  void **gather** (const U &u, std::vector< U > &vU) throw ()
- template<class U >
  void **all_gather** (const U &u, std::vector< U > &vU) throw ()
- template<class U >
  void **scatter** (U &u) throw ()
- template<class U >
  void **scatter** (const std::vector< U > &vU, U &u) throw ()

Static Public Member Functions

- static bool **readTag** (abstractCommHandle ∗h, int &tag) throw ()
- static bool **writeTag** (abstractCommHandle ∗h, int tag) throw ()
- static processCommHandle ∗ **open** (const int rank, const MPI::Comm &comm, const char ∗mode, int tag=0, TRANSFER_KIND transfer_-type=POINT2POINT) throw ()
- static void **attachMPISendBuffer** (size_t bufSize=32 ∗1024 ∗1024) throw ()
- static void **detachMPISendBuffer** (void) throw ()
- static bool **MPIbuffered** (void)

Protected Member Functions

- virtual bool **transferBufferHeader** (void) const
- virtual bool **partialBufferTransfer** (void) const
- **processCommHandle** (int rank, const MPI::Comm &comm, int tag, TRANSFER_KIND transfer_kind)

1.7.8 distributed_solver_module::distributedSolver_module Class Reference

Classes

- struct parameters

Public Member Functions

- const Epetra_Comm & **comm** (void) const

- void **global_block_indices** (size_t &global_row_blocks, std::vector< size_t > &v_rb, size_t &global_col_blocks, std::vector< size_t > &v_cb) const throw ()
- void **init_LHS_block** (const size_t rb, const size_t cb, const std::complex< double > *M) throw ()
- void **init_RHS_block** (const size_t rb, const std::complex< double > *M) throw ()
- void **clear_X_block** (const size_t rb) throw ()
- void **factor** (const parameters &param) throw ()
- void **solve** (const parameters &param) throw ()
- void **solve_masked** (const size_t global_row_blocks, const std::vector< size_t > &v_rb, const size_t global_col_blocks, const std::vector< size_t > &v_cb, const std::vector< size_t > &vn_RHS_cols, const parameters &param) throw ()
- size_t **max_local_row_blocks** (void) const throw ()
- void **get_LHS_block** (const size_t rb, const size_t cb, std::complex< double > *M) const throw ()
- void **get_RHS_block** (const size_t rb, std::complex< double > *M) const throw ()
- void **get_X_block** (const size_t rb, std::complex< double > *M) const throw ()
- void **get_X_columns** (const size_t rb, const std::vector< size_t > &vn_RHS_cols, std::complex< double > *M, const std::vector< size_t > &vn_dest_cols) const throw ()
- void **init** (size_t row_blocks, size_t col_blocks, size_t block_nrows, size_t block_ncols, size_t N_RHS, const MPI_Comm &comm) throw ()

- void **clear** (void)

1.7.9 distributed_solver_module::distributedSolver_module::parameters Struct Reference

Public Member Functions

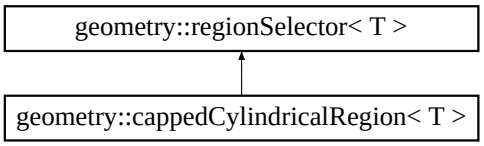
- **parameters** (double convergence_tolerance_, size_t max_iterations_, size_t max_restarts_, size_t RHS_blocks_, size_t krylov_dim_, int verbosity_)

Public Attributes

- double **convergence_tolerance**
- size_t **max_iterations**
- size_t **max_restarts**
- size_t **RHS_blocks**
- size_t **krylov_dim**
- int **verbosity**

1.7.10 geometry::cappedCylindricalRegion< T > Class Template Reference

Inheritance diagram for geometry::cappedCylindricalRegion< T >:



Public Types

- typedef regionSelector< T >::magnitudeType **magnitudeType**

Public Member Functions

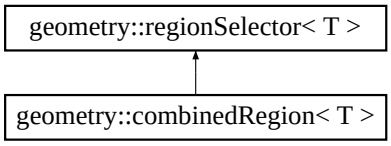
- virtual regionSelector< T >::region_kind **kind** (void) const
- const T & **a** (void) const
- const T & **b** (void) const
- virtual magnitudeType **circumscribingRadius** (void) const throw ()
- virtual magnitudeType **circumscribedRadius** (void) const throw ()
- virtual magnitudeType **volume** (void) const throw ()
- virtual T **scale** (void) const
- virtual void **rescale** (const T &newScale) throw ()
- virtual void **rescalePhysical** (const magnitudeType &newScale) throw ()
- virtual regionSelector< T > * **clone** (void) const
- virtual T **r** (const T &theta, const T &phi, bool cosTheta=false) const throw ()
- virtual void **r** (const std::vector< T > &vTheta, const std::vector< T > &vPhi, std::vector< T > &vR, std::vector< T > &vN_m, std::vector< T > &vN_0, std::vector< T > &vN_p, bool cosTheta=false) const throw ()
- virtual void **r** (const std::vector< T > &vTheta, const long m, std::vector< T > &vR, std::vector< T > &vN_m, std::vector< T > &vN_0, std::vector< T > &vN_p, bool cosTheta=false) const throw ()
- virtual void **n** (const std::vector< T > &vTheta, const std::vector< T > &vPhi, std::vector< T > &vR, std::vector< T > &vN_r, std::vector< T > &vN_t, std::vector< T > &vN_p, bool cosTheta=false) const throw ()

- virtual void **get_vM_surface** (std::vector< long > &vM) const
- virtual symmetry_kind **symmetry** (spherical_coordinate eCoord) const
- virtual bool **readBinary** (abstractCommHandle *fp)
- virtual bool **writeBinary** (abstractCommHandle *fp) const
- cappedCylindricalRegion & **operator=** (const cappedCylindricalRegion &other)
- **cappedCylindricalRegion** (const cappedCylindricalRegion &other)
- **cappedCylindricalRegion** (const magnitudeType &r0, const magnitudeType &t0, const magnitudeType &p0, const T &a__, const T &b__, const T &a_ext, bool interior=true)
- **cappedCylindricalRegion** (const magnitudeType &x0, const magnitudeType &y0, const magnitudeType &z0, const T &a__, const T &b__, const T &a_ext, bool interior, bool)

template<class T> class geometry::cappedCylindricalRegion< T >

1.7.11 geometry::combinedRegion< T > Class Template Reference

Inheritance diagram for geometry::combinedRegion< T >:



Public Types

- enum **REGION_OPERATOR** { **NOP**, **UNION**, **INTERSECTION** }
- typedef regionSelector< T >::magnitudeType **magnitudeType**

Public Member Functions

- virtual regionSelector< T >::region_kind **kind** (void) const
- virtual bool **inRegion** (const T &r, const T &t, const T &p) const throw ()
- virtual magnitudeType **circumscribingRadius** (void) const throw ()
- virtual magnitudeType **circumscribedRadius** (void) const throw ()
- virtual magnitudeType **volume** (void) const throw ()
- virtual T **scale** (void) const
- virtual void **invertPosition** (void)
- virtual void **rescale** (const T &newScale) throw ()
- virtual void **rescalePhysical** (const magnitudeType &newScale) throw ()
- virtual regionSelector< T > * **clone** (void) const
- virtual bool **readBinary** (abstractCommHandle *fp)
- virtual bool **writeBinary** (abstractCommHandle *fp) const
- combinedRegion< T > & **operator=** (const combinedRegion< T > &other)

- **combinedRegion** (const combinedRegion< T > &other)
- **combinedRegion** (const regionSelector< T > &regionA, const regionSelector< T > &regionB, REGION_OPERATOR eOp, T scale=one< T >())

template<class T> class geometry::combinedRegion< T >

1.7.12 geometry::icosphere< T > Class Template Reference

Public Types

- typedef numberTraits< T >::magnitudeType **R**
- typedef std::vector< ntuple< R, 3 > > **pointList**

Static Public Member Functions

- static void **init** (const R &r, size_t N, pointList &dest) throw ()

template<class T> class geometry::icosphere< T >

1.7.13 geometry::integerLattice Class Reference

Static Public Member Functions

- static void **balancedLattice** (size_t N, const size_t DIM, std::vector< ntuple< long, 3 > > &vDest) throw ()

Static Protected Member Functions

- static size_t **N_permutations** (const ntuple< long, 3 > &p, const size_t DIM, bool mirror) throw ()
- static void **permute** (const ntuple< long, 3 > &pSrc, const size_t DIM, std::vector< ntuple< long, 3 > > &vDest, bool mirror) throw ()

- static void **rawPermute** (const ntuple< long, 3 > &p, const size_t DIM, std::vector< ntuple< long, 3 > > &vDest) throw ()
- static void **mirrorPermute** (const ntuple< long, 3 > &p, const size_t DIM, std::vector< ntuple< long, 3 > > &vDest) throw ()
- static void **maximizeAdjacentDistance** (std::vector< ntuple< long, 3 > > &vP) throw ()

1.7.14 geometry::lattice< T > Class Template Reference

Classes

- struct distance_to_point
- struct less_radius
- class material_info
- class particle_info

Public Types

- typedef T **C**
- typedef numberTraits< T >::magnitudeType **R**
- typedef numberTraits< T >::integralType **Z**
- typedef std::vector< ntuple< R, 3 > > **ntupleList**
- typedef std::vector< particle_info > **particleList**

Static Public Member Functions

- static void **read** (std::istream &is, particleList &dest) throw ()

- static void **write** (std::ostream &os, const particleList &src) throw ()
- static void **scale** (const R &f, const particleList &src, particleList &dest) throw ()
- static void **moveCentroid** (const ntuple< R, 3 > &pC, particleList &vP) throw ()
- static void **translate** (const particleList &vP0, const ntuple< R, 3 > &pC, particleList &vDest) throw ()
- static void **rotate** (const ntuple< R, 3 > &eulerAngles, particleList &vP) throw ()
- static void **concat** (const particleList &vP1, const particleList &vP2, particleList &vDest) throw ()
- static void **chain** (const ntuple< R, 3 > &a_0, size_t N, ntupleList &dest) throw ()
- static void **disk** (const ntuple< R, 3 > &a_0, const ntuple< R, 3 > &a_1, size_t N, ntupleList &dest) throw ()
- static void **ball** (const ntuple< R, 3 > &a_0, const ntuple< R, 3 > &a_1, const ntuple< R, 3 > &a_2, size_t N, ntupleList &dest) throw ()
- static void **slab** (const ntuple< R, 3 > &a_0, const ntuple< R, 3 > &a_1, const ntuple< R, 3 > &a_2, const R &width, const R &depth, const R &height, ntupleList &dest) throw ()
- static void **chain** (const ntuple< R, 3 > &a_0, const regionSelector< R > &region, ntupleList &dest, const R &alpha=zero< R >(), const R &beta=zero< R >(), const R &gamma=zero< R >()) throw ()
- static void **disk** (const ntuple< R, 3 > &a_0, const ntuple< R, 3 > &a_1, const regionSelector< R > &region, ntupleList &dest, const R

&alpha=zero< R >(), const R &beta=zero< R >(), const R &gamma=zero< R >()) throw ()

- static void **ball** (const ntuple< R, 3 > &a_0, const ntuple< R, 3 > &a_1, const ntuple< R, 3 > &a_2, const regionSelector< R > &region, ntupleList &dest, const R &alpha=zero< R >(), const R &beta=zero< R >(), const R &gamma=zero< R >()) throw ()
- static void **disk** (const ntuple< R, 3 > &a_0, const ntuple< R, 3 > &a_-1, size_t N, const regionSelector< R > &region, ntupleList &dest, const R &alpha=zero< R >(), const R &beta=zero< R >(), const R &gamma=zero< R >()) throw ()
- static void **ball** (const ntuple< R, 3 > &a_0, const ntuple< R, 3 > &a_1, const ntuple< R, 3 > &a_2, size_t N, const regionSelector< R > &region, ntupleList &dest, const R &alpha=zero< R >(), const R &beta=zero< R >(), const R &gamma=zero< R >()) throw ()
- static void **linearChain** (const R &a, size_t N, ntupleList &dest, const R &alpha=zero< R >(), const R &beta=zero< R >(), const R &gamma=zero< R >()) throw ()
- static void **hcpBasis** (const R &a, ntuple< R, 3 > &a_0, ntuple< R, 3 > &a_1, ntuple< R, 3 > &a_2, const R &alpha=zero< R >(), const R &beta=zero< R >(), const R &gamma=zero< R >()) throw ()
- static void **scBasis** (const R &a, ntuple< R, 3 > &a_0, ntuple< R, 3 > &a_1, ntuple< R, 3 > &a_2, const R &alpha=zero< R >(), const R &beta=zero< R >(), const R &gamma=zero< R >()) throw ()
- static void **hcpDisk** (const R &a, size_t N, ntupleList &dest, const R &alpha=zero< R >(), const R &beta=zero< R >(), const R &gamma=zero< R >()) throw ()

- static void **hcpBall** (const R &a, size_t N, ntupleList &dest, const R &alpha=zero< R >(), const R &beta=zero< R >(), const R &gamma=zero< R >()) throw ()
- static void **scDisk** (const R &a, size_t N, ntupleList &dest, const R &alpha=zero< R >(), const R &beta=zero< R >(), const R &gamma=zero< R >()) throw ()
- static void **scBall** (const R &a, size_t N, ntupleList &dest, const R &alpha=zero< R >(), const R &beta=zero< R >(), const R &gamma=zero< R >()) throw ()
- static void **scale** (const R &f, const ntupleList &src, ntupleList &dest) throw ()
- static ntuple< R, 3 > **centroid** (const ntupleList &vP) throw ()
- static void **zeroCentroid** (ntupleList &vP) throw ()
- static void **moveCentroid** (const ntuple< R, 3 > &pC, ntupleList &vP) throw ()
- static void **translate** (const ntupleList &vP0, const ntuple< R, 3 > &pC, ntupleList &vDest) throw ()
- static void **rotate** (const ntuple< R, 3 > &eulerAngles, ntupleList &vP) throw ()
- static void **concat** (const ntupleList &vP1, const ntupleList &vP2, ntupleList &vDest) throw ()
- static void **sort** (const ntupleList &src, ntupleList &dest) throw ()
- static void **neighborhood_lists** (const ntupleList &src, std::vector< std::vector< size_t > > &vvnNeighborhoods) throw ()
- static void **randomize** (const R &r, const R &a, const R &variance, const ntupleList &src, ntupleList &dest) throw ()

- static void **cull** (const ntupleList &src, const R &cullFraction, ntupleList &dest) throw ()
- static void **exclude_overlap** (const ntupleList &src1, const R &r1, const ntupleList &src2, const R &r2, ntupleList &dest) throw ()
- static bool **overlapTest** (const R &r, const ntupleList &src) throw ()
- static bool **overlapTest** (const R &r, typename ntupleList::const_iterator itStart, typename ntupleList::const_iterator itEnd) throw ()
- static void **read** (const std::string &fileName, ntupleList &dest) throw ()
- static void **write** (const ntupleList &src, const std::string &fileName) throw ()
- static void **read** (const std::string &fileName, const material_info &default_material, const R &default_radius, const ntuple< R, 3 > &default_angle, particleList &dest) throw ()
- static void **write** (const particleList &src, const std::string &fileName) throw ()
- static void **writePOV** (const ntupleList &points, const R &radius, const std::string &sPOV_fileName, const std::string &sPOV_header="", const std::string &sPOV_footer="") throw ()
- static void **writePOV** (const particleList &particles, const R &radius, const std::string &sPOV_fileName, const std::string &sPOV_header="", const std::string &sPOV_footer="") throw ()

Static Protected Member Functions

- static R **unitCellArea** (const ntuple< R, 3 > &a_0, const ntuple< R, 3 > &a_1)

- static R **unitCellVolume** (const ntuple< R, 3 > &a_0, const ntuple< R, 3 > &a_1, const ntuple< R, 3 > &a_2)
- static void **fill1D** (const ntuple< R, 3 > &a_0, const regionSelector< R > &region, ntupleList &dest) throw ()
- static void **fill2D** (const ntuple< R, 3 > &a_0, const ntuple< R, 3 > &a_1, const regionSelector< R > &region, ntupleList &dest) throw ()
- static void **fill3D** (const ntuple< R, 3 > &a_0, const ntuple< R, 3 > &a_1, const ntuple< R, 3 > &a_2, const regionSelector< R > &region, ntupleList &dest) throw ()

Static Protected Attributes

- static const size_t **maxIterations** = 100000

template<class T> class geometry::lattice< T >

1.7.15 geometry::lattice< T >::distance_to_point Struct Reference

Public Member Functions

- bool **operator()** (size_t n1, size_t n2)
- **distance_to_point** (const ntupleList &src, const ntuple< R, 3 > &refPoint)

template<class T> struct geometry::lattice< T >::distance_to_point

1.7.16 geometry::lattice< T >::less_radius Struct Reference

Public Member Functions

- bool **operator()** (const ntuple< R, 3 > &p1, const ntuple< R, 3 > &p2)

template<class T> struct geometry::lattice< T >::less_radius

1.7.17 geometry::lattice< T >::material_info Class Reference

Public Types

- typedef python_util::extensible_parameters_base< T > **base_class**
- typedef T **C**
- typedef numberTraits< T >::magnitudeType **R**
- typedef numberTraits< T >::integralType **Z**

Public Member Functions

- bool **readBinary** (abstractCommHandle *fp) throw ()
- bool **writeBinary** (abstractCommHandle *fp) const throw ()
- size_t **binarySize** (void) const throw ()
- void **read** (std::istream &is) throw ()
- void **write** (std::ostream &os) const throw ()
- void **swap** (material_info &other)
- void **extract** (const python_util::simple_object_base *pobject) throw ()
- virtual void **clone_from** (const python_util::simple_object_base *pobject, bool transfer_ownership=false) throw ()

- virtual python_util::extensible_parameters_base< T > * **clone** (void) const throw ()
- void **copy** (const material_info &other)
- material_info & **operator=** (const material_info &other)
- **material_info** (const material_info &other)
- **material_info** (const std::string &name_)
- template<class U >
  material_info (const std::string &name_, const U &coeff_)
- **material_info** (const python_util::simple_object_base *pobject, bool transfer_ownership=false) throw ()

template<class T> class geometry::lattice< T >::material_info

1.7.18 geometry::lattice< T >::particle_info Class Reference

Public Types

- typedef python_util::extensible_parameters_base< T > **base_class**
- typedef base_class::object_list **object_list**
- typedef base_class::object_map **object_map**
- typedef T **C**
- typedef numberTraits< T >::magnitudeType **R**
- typedef numberTraits< T >::integralType **Z**

Public Member Functions

- R & **radius** (void)

- const R & **radius** (void) const
- bool **readBinary** (abstractCommHandle ∗fp) throw ()
- bool **writeBinary** (abstractCommHandle ∗fp) const throw ()
- size_t **binarySize** (void) const throw ()
- void **read** (std::istream &is) throw ()
- void **write** (std::ostream &os) const throw ()
- void **extract** (const python_util::simple_object_base ∗pobject) throw ()
- python_util::simple_object_base ∗ **repn_virtual** (void) const
- void **swap** (particle_info &other)
- virtual void **clone_from** (const python_util::simple_object_base ∗pobject, bool transfer_ownership=false) throw ()
- virtual python_util::extensible_parameters_base< T > ∗ **clone** (void) const throw ()
- void **copy** (const particle_info &other)
- particle_info & **operator=** (const particle_info &other)
- **particle_info** (const particle_info &other)
- **particle_info** (const material_info &material_, const R &radius_, const ntuple< R, 3 > &position_, const ntuple< R, 3 > &angle_)
- **particle_info** (const python_util::simple_object_base ∗pobject, bool transfer_ownership=false) throw ()

Public Attributes

- material_info **material**
- ntuple< R, 3 > **position**
- ntuple< R, 3 > **angle**

template<class T> class geometry::lattice< T >::particle_info

1.7.19 geometry::rectangularPrism< T > Class Template Reference

Public Types

- typedef numberTraits< T >::magnitudeType **R**

Public Member Functions

- void **vertex** (const long &i1, const long &i2, const long &i3, T &t1, T &t2, T &t3) const throw ()
- void **sphereDomain** (T &rMin, T &rMax, T &tMin, T &tMax, T &pMin, T &pMax, size_t nX, size_t nY, size_t nZ) const throw ()
- bool **projectionIncludesCut** (void) const
- void **write** (std::ostream &os) const
- **rectangularPrism** (const T &xMin, const T &xMax, const T &yMin, const T &yMax, const T &zMin, const T &zMax)

Static Public Member Functions

- static bool **inDomainInclusive** (const T &x, const T &a, const T &b)
- static bool **inDomainExclusive** (const T &x, const T &a, const T &b)

Protected Member Functions

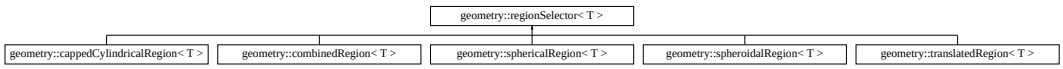
- T **r_s** (const T &s, const T &x0, const T &y0, const T &z0, const T &x1, const T &y1, const T &z1) const

- T **t_s** (const T &s, const T &x0, const T &y0, const T &z0, const T &x1, const T &y1, const T &z1) const
- T **p_s** (const T &s, const T &x0, const T &y0, const T &z0, const T &x1, const T &y1, const T &z1) const
- void **lineSegmentDomain** (const long &v0, const long &v1, T &rMin, T &rMax, T &tMin, T &tMax, T &pMin, T &pMax, bool reflect_x_axis=false) const throw ()
- bool **faceZero** (const long &faceBit, bool max_, T &rMin) const throw ()
- void **faceIndices** (const long &faceBit, bool max_, long &base, long &left, long &right, long &para) const throw ()
- void **vertex** (const long &index, T &t1, T &t2, T &t3) const throw ()

template<class T> class geometry::rectangularPrism< T >

1.7.20 geometry::regionSelector< T > Class Template Reference

Inheritance diagram for geometry::regionSelector< T >:



Public Types

- enum **region_kind** {

  ABSTRACT_REGION, **SPHERICAL_REGION**, **SPHEROIDAL_-REGION**, **CAPPED_CYLINDRICAL_REGION**,

COMBINED_REGION, **TRANSLATED_REGION** }

- typedef numberTraits< T >::magnitudeType **magnitudeType**

Public Member Functions

- virtual region_kind **kind** (void) const
- void **complementSelf** (void)
- bool **interior** (void) const
- virtual bool **inRegion** (const T &r, const T &t, const T &p) const throw ()
- virtual magnitudeType **circumscribingRadius** (void) const throw ()
- virtual magnitudeType **circumscribedRadius** (void) const throw ()
- virtual magnitudeType **volume** (void) const throw ()
- virtual T **scale** (void) const
- virtual void **invertPosition** (void)
- virtual void **rescale** (const T &newScale) throw ()
- virtual void **rescalePhysical** (const magnitudeType &newScale) throw ()
- void **recenter** (const magnitudeType &r0, const magnitudeType &t0, const magnitudeType &p0)
- void **recenter** (const magnitudeType &x0__, const magnitudeType &y0__, const magnitudeType &z0__, bool)
- virtual regionSelector< T > * **clone** (void) const
- const magnitudeType & **x0** (void) const
- const magnitudeType & **y0** (void) const
- const magnitudeType & **z0** (void) const
- virtual T **r** (const T &theta, const T &phi, bool cosTheta=false) const throw ()

- virtual void **r** (const std::vector< T > &vTheta, const std::vector< T > &vPhi, std::vector< T > &vR, std::vector< T > &vN_m, std::vector< T > &vN_0, std::vector< T > &vN_p, bool cosTheta=false) const throw ()
- virtual void **r** (const std::vector< T > &vTheta, const long m, std::vector< T > &vR, std::vector< T > &vN_m, std::vector< T > &vN_0, std::vector< T > &vN_p, bool cosTheta=false) const throw ()
- virtual void **n** (const std::vector< T > &vTheta, const std::vector< T > &vPhi, std::vector< T > &vR, std::vector< T > &vN_r, std::vector< T > &vN_t, std::vector< T > &vN_p, bool cosTheta=false) const throw ()
- virtual void **get_vM_surface** (std::vector< long > &vM) const
- virtual symmetry_kind **symmetry** (spherical_coordinate eCoord) const
- void **separateMesh** (const std::vector< T > &vR, const std::vector< T > &vT, const std::vector< T > &vP, const std::vector< unsigned long > &vnRR, const std::vector< unsigned long > &vnTT, const std::vector< unsigned long > &vnPP, std::vector< unsigned long > &vnRR_sub, std::vector< unsigned long > &vnTT_sub, std::vector< unsigned long > &vnPP_sub) const
- void **separateMesh** (const ntuple< magnitudeType, 3 > &center, const std::vector< T > &vR, const std::vector< T > &vT, const std::vector< T > &vP, const std::vector< unsigned long > &vnRR, const std::vector< unsigned long > &vnTT, const std::vector< unsigned long > &vnPP, std::vector< unsigned long > &vnRR_sub, std::vector< unsigned long > &vnTT_sub, std::vector< unsigned long > &vnPP_sub) const
- const std::vector< unsigned long > & **sub2SuperMesh** (void) const
- void **combineMesh** (const std::vector< T > &vSub, std::vector< T > &vSuper, bool accumulate=false) const

- virtual bool **readBinary** (abstractCommHandle ∗fp)
- virtual bool **writeBinary** (abstractCommHandle ∗fp) const
- regionSelector & **operator=** (const regionSelector &other)
- **regionSelector** (const regionSelector &other)
- **regionSelector** (bool interior, const magnitudeType &x0, const magnitudeType &y0, const magnitudeType &z0)

Static Public Member Functions

- static bool **readBinaryVirtual** (abstractCommHandle ∗fp, regionSelector< T > ∗&pregion)
- static bool **writeBinaryVirtual** (abstractCommHandle ∗fp, const regionSelector< T > ∗pregion)

Protected Member Functions

- void **init** (bool interior, const magnitudeType &x0, const magnitudeType &y0, const magnitudeType &z0)

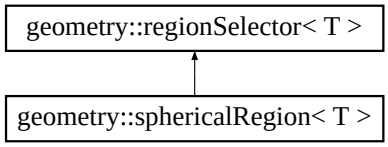
Protected Attributes

- magnitudeType **x0_**
- magnitudeType **y0_**
- magnitudeType **z0_**

template<class T> class geometry::regionSelector< T >

1.7.21 geometry::sphericalRegion< T > Class Template Reference

Inheritance diagram for geometry::sphericalRegion< T >:



Public Types

- typedef regionSelector< T >::magnitudeType **magnitudeType**

Public Member Functions

- virtual regionSelector< T >::region_kind **kind** (void) const
- virtual magnitudeType **circumscribingRadius** (void) const throw ()
- virtual magnitudeType **circumscribedRadius** (void) const throw ()
- virtual magnitudeType **volume** (void) const throw ()
- virtual T **scale** (void) const
- virtual void **rescale** (const T &newScale) throw ()
- virtual void **rescalePhysical** (const magnitudeType &newScale) throw ()
- virtual regionSelector< T > * **clone** (void) const
- virtual T **r** (const T &theta, const T &phi, bool cosTheta=false) const throw ()
- virtual void **r** (const std::vector< T > &vTheta, const std::vector< T > &vPhi, std::vector< T > &vR, std::vector< T > &vN_m, std::vector< T > &vN_0, std::vector< T > &vN_p, bool cosTheta=false) const throw ()

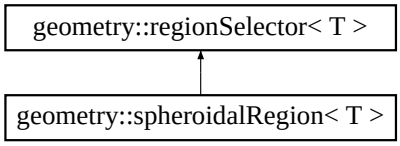
- virtual void **r** (const std::vector< T > &vTheta, const long m, std::vector< T > &vR, std::vector< T > &vN_m, std::vector< T > &vN_0, std::vector< T > &vN_p, bool cosTheta=false) const throw ()
- virtual void **n** (const std::vector< T > &vTheta, const std::vector< T > &vPhi, std::vector< T > &vR, std::vector< T > &vN_r, std::vector< T > &vN_t, std::vector< T > &vN_p, bool cosTheta=false) const throw ()
- virtual void **get_vM_surface** (std::vector< long > &vM) const
- virtual symmetry_kind **symmetry** (spherical_coordinate eCoord) const
- virtual bool **readBinary** (abstractCommHandle *fp)
- virtual bool **writeBinary** (abstractCommHandle *fp) const
- sphericalRegion & **operator=** (const sphericalRegion &other)
- **sphericalRegion** (const sphericalRegion &other)
- **sphericalRegion** (const magnitudeType &r0, const magnitudeType &t0, const magnitudeType &p0, const T &r, const T &r_ext, bool interior=true)

- **sphericalRegion** (const magnitudeType &x0, const magnitudeType &y0, const magnitudeType &z0, const T &r, const T &r_ext, bool interior, bool)

template<class T> class geometry::sphericalRegion< T >

1.7.22 geometry::spheroidalRegion< T > Class Template Reference

Inheritance diagram for geometry::spheroidalRegion< T >:



Public Types

- typedef regionSelector< T >::magnitudeType **magnitudeType**

Public Member Functions

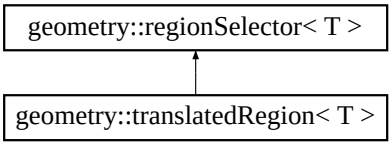
- virtual regionSelector< T >::region_kind **kind** (void) const
- const T & **a** (void)
- const T & **b** (void)
- virtual magnitudeType **circumscribingRadius** (void) const throw ()
- virtual magnitudeType **circumscribedRadius** (void) const throw ()
- virtual magnitudeType **volume** (void) const throw ()
- virtual T **scale** (void) const
- virtual void **rescale** (const T &newScale) throw ()
- virtual void **rescalePhysical** (const magnitudeType &newScale) throw ()
- virtual regionSelector< T > * **clone** (void) const
- virtual T **r** (const T &theta, const T &phi, bool cosTheta=false) const throw ()
- virtual void **r** (const std::vector< T > &vTheta, const std::vector< T > &vPhi, std::vector< T > &vR, std::vector< T > &vN_m, std::vector< T > &vN_0, std::vector< T > &vN_p, bool cosTheta=false) const throw ()

- virtual void **r** (const std::vector< T > &vTheta, const long m, std::vector< T > &vR, std::vector< T > &vN_m, std::vector< T > &vN_0, std::vector< T > &vN_p, bool cosTheta=false) const throw ()
- virtual void **n** (const std::vector< T > &vTheta, const std::vector< T > &vPhi, std::vector< T > &vR, std::vector< T > &vN_r, std::vector< T > &vN_t, std::vector< T > &vN_p, bool cosTheta=false) const throw ()
- virtual void **get_vM_surface** (std::vector< long > &vM) const
- virtual symmetry_kind **symmetry** (spherical_coordinate eCoord) const
- virtual bool **readBinary** (abstractCommHandle *fp)
- virtual bool **writeBinary** (abstractCommHandle *fp) const
- spheroidalRegion & **operator=** (const spheroidalRegion &other)
- **spheroidalRegion** (const spheroidalRegion &other)
- **spheroidalRegion** (const magnitudeType &r0, const magnitudeType &t0, const magnitudeType &p0, const T &a__, const T &b__, const T &a_ext, bool interior=true)
- **spheroidalRegion** (const magnitudeType &x0, const magnitudeType &y0, const magnitudeType &z0, const T &a__, const T &b__, const T &a_ext, bool interior, bool)

template<class T> class geometry::spheroidalRegion< T >

1.7.23 geometry::translatedRegion< T > Class Template Reference

Inheritance diagram for geometry::translatedRegion< T >:



Public Types

- typedef regionSelector< T >::magnitudeType **magnitudeType**

Public Member Functions

- virtual regionSelector< T >::region_kind **kind** (void) const
- const regionSelector< T > & **region** (void) const
- const magnitudeType & **alpha** (void) const
- const magnitudeType & **beta** (void) const
- const magnitudeType & **gamma** (void) const
- virtual bool **inRegion** (const T &r, const T &t, const T &p) const throw ()
- virtual magnitudeType **circumscribingRadius** (void) const throw ()
- virtual magnitudeType **circumscribedRadius** (void) const throw ()
- virtual magnitudeType **volume** (void) const throw ()
- virtual T **scale** (void) const
- virtual void **invertPosition** (void)
- virtual void **rescale** (const T &newScale) throw ()
- virtual void **rescalePhysical** (const magnitudeType &newScale) throw ()
- virtual void **n** (const std::vector< T > &vTheta, const std::vector< T > &vPhi, std::vector< T > &vR, std::vector< T > &vN_r, std::vector< T > &vN_t, std::vector< T > &vN_p, bool cosTheta=false) const throw ()

- virtual regionSelector< T > ∗ **clone** (void) const
- virtual bool **readBinary** (abstractCommHandle ∗fp)
- virtual bool **writeBinary** (abstractCommHandle ∗fp) const
- translatedRegion< T > & **operator=** (const translatedRegion< T > &other)

- **translatedRegion** (const translatedRegion< T > &other)
- **translatedRegion** (const regionSelector< T > &region, const magnitudeType &x0, const magnitudeType &y0, const magnitudeType &z0, const magnitudeType &alpha, const magnitudeType &beta, const magnitudeType &gamma)

template<class T> class geometry::translatedRegion< T >

1.7.24 gmm::linalg_traits< const indefiniteSparseMatrix< T > > Struct Template Reference

Public Types

- typedef const indefiniteSparseMatrix< T > **this_type**
- typedef this_type::key_type **key_type**
- typedef indefiniteSparseMatrix< T > **origin_type**
- typedef const origin_type ∗ **porigin_type**
- typedef linalg_false **is_reference**
- typedef abstract_matrix **linalg_type**
- typedef T **value_type**
- typedef this_type::reference **reference**

- typedef this_type::const_reference **const_reference**
- typedef abstract_sparse **storage_type**
- typedef sparseNDimensional_iterator< origin_type, 2, 2 > **row_iterator**
- typedef sparseNDimensional_iterator< const origin_type, 2, 2 > **const_row_iterator**
- typedef sparseSubdimensionReference< origin_type, 1, 2 > **sub_row_type**
- typedef sparseSubdimensionReference< const origin_type, 1, 2 > **const_sub_row_type**
- typedef sub_row_type::iterator **col_iterator**
- typedef sub_row_type::const_iterator **const_col_iterator**
- typedef abstract_null_type **sub_col_type**
- typedef abstract_null_type **const_sub_col_type**
- typedef row_major **sub_orientation**
- typedef linalg_false **index_sorted**

Static Public Member Functions

- static size_type **nrows** (const this_type &m)
- static size_type **ncols** (const this_type &m)
- static const_sub_row_type **row** (const const_row_iterator &it)
- static const_row_iterator **row_begin** (const this_type &m)
- static const_row_iterator **row_end** (const this_type &m)
- static porigin_type **origin** (this_type &m)
- static void **do_clear** (this_type &m)
- static const_reference **access** (const const_col_iterator &itcol, size_type j)

- static void **resize** (this_type &v, size_type m, size_type n)
- static void **reshape** (this_type &v, size_type m, size_type n)

template<typename T> struct gmm::linalg_traits< const indefiniteSparseMatrix< T > >

1.7.25 gmm::linalg_traits< const indefiniteVector< T > > Struct Template Reference

Public Types

- typedef const indefiniteVector< T > **this_type**
- typedef indefiniteVector< T > **origin_type**
- typedef linalg_false **is_reference**
- typedef abstract_vector **linalg_type**
- typedef T **value_type**
- typedef ref_elt_vector< T, indefiniteVector< T > > **reference**
- typedef this_type::iterator **iterator**
- typedef this_type::const_iterator **const_iterator**
- typedef abstract_sparse **storage_type**
- typedef linalg_false **index_sorted**

Static Public Member Functions

- static size_type **size** (const this_type &v)
- static const_iterator **begin** (const this_type &v)
- static const_iterator **end** (const this_type &v)

- static const origin_type ∗ **origin** (const this_type &v)
- static void **clear** (origin_type ∗o, const iterator &, const iterator &)
- static void **do_clear** (this_type &v)
- static value_type **access** (const origin_type ∗o, const const_iterator &, const const_iterator &, size_type i)
- static void **resize** (this_type &v, size_type n)

template<typename T> struct gmm::linalg_traits< const indefiniteVector< T > >

1.7.26 gmm::linalg_traits< indefiniteSparseMatrix< T > > Struct Template Reference

Public Types

- typedef indefiniteSparseMatrix< T > **this_type**
- typedef this_type::key_type **key_type**
- typedef this_type **origin_type**
- typedef origin_type ∗ **porigin_type**
- typedef linalg_false **is_reference**
- typedef abstract_matrix **linalg_type**
- typedef T **value_type**
- typedef this_type::reference **reference**
- typedef this_type::const_reference **const_reference**
- typedef abstract_sparse **storage_type**
- typedef sparseNDimensional_iterator< this_type, 2, 2 > **row_iterator**

- typedef sparseNDimensional_iterator< const this_type, 2, 2 > **const_row_-iterator**
- typedef sparseSubdimensionReference< this_type, 1, 2 > **sub_row_type**
- typedef sparseSubdimensionReference< const this_type, 1, 2 > **const_sub_-row_type**
- typedef sub_row_type::iterator **col_iterator**
- typedef sub_row_type::const_iterator **const_col_iterator**
- typedef abstract_null_type **sub_col_type**
- typedef abstract_null_type **const_sub_col_type**
- typedef row_major **sub_orientation**
- typedef linalg_false **index_sorted**

Static Public Member Functions

- static size_type **nrows** (const this_type &m)
- static size_type **ncols** (const this_type &m)
- static const_sub_row_type **row** (const const_row_iterator &it)
- static sub_row_type **row** (const row_iterator &it)
- static row_iterator **row_begin** (this_type &m)
- static row_iterator **row_end** (this_type &m)
- static const_row_iterator **row_begin** (const this_type &m)
- static const_row_iterator **row_end** (const this_type &m)
- static origin_type * **origin** (this_type &m)
- static const origin_type * **origin** (const this_type &m)
- static void **do_clear** (this_type &m)
- static reference **access** (const col_iterator &itcol, size_type j)

- static void **resize** (this_type &v, size_type m, size_type n)
- static void **reshape** (this_type &v, size_type m, size_type n)

template<**typename T**> **struct gmm::linalg_traits**< **indefiniteSparseMatrix**< **T** > >

1.7.27 gmm::linalg_traits< indefiniteVector< T > > Struct Template Reference

Public Types

- typedef indefiniteVector< T > **this_type**
- typedef this_type **origin_type**
- typedef linalg_false **is_reference**
- typedef abstract_vector **linalg_type**
- typedef T **value_type**
- typedef ref_elt_vector< T, indefiniteVector< T > > **reference**
- typedef this_type::const_reference **const_reference**
- typedef this_type::iterator **iterator**
- typedef this_type::const_iterator **const_iterator**
- typedef abstract_sparse **storage_type**
- typedef linalg_false **index_sorted**

Static Public Member Functions

- static size_type **size** (const this_type &v)
- static iterator **begin** (this_type &v)

- static const_iterator **begin** (const this_type &v)
- static iterator **end** (this_type &v)
- static const_iterator **end** (const this_type &v)
- static origin_type * **origin** (this_type &v)
- static const origin_type * **origin** (const this_type &v)
- static void **clear** (origin_type *o, const iterator &, const iterator &)
- static void **do_clear** (this_type &v)
- static value_type **access** (const origin_type *o, const const_iterator &, const const_iterator &, size_type i)
- static reference **access** (origin_type *o, const iterator &, const iterator &, size_type i)
- static void **resize** (this_type &v, size_type n)

template<typename T> struct gmm::linalg_traits< indefiniteVector< T > >

1.7.28 gmm::linalg_traits< sparseSubdimensionReference< const VECT, DIM, NDIM > > Struct Template Reference

Public Types

- typedef sparseSubdimensionReference< const VECT, DIM, NDIM > **this_type**
- typedef this_type::origin_type **origin_type**
- typedef this_type::porigin_type **porigin_type**
- typedef which_reference< VECT * >::is_reference **is_reference**
- typedef abstract_vector **linalg_type**

- typedef VECT::value_type **value_type**
- typedef this_type::reference **reference**
- typedef this_type::iterator **iterator**
- typedef this_type::const_iterator **const_iterator**
- typedef abstract_sparse **storage_type**
- typedef linalg_false **index_sorted**

Static Public Member Functions

- static size_type **size** (const this_type &v)
- static const_iterator **begin** (const this_type &v)
- static const_iterator **end** (const this_type &v)
- static porigin_type **origin** (this_type &v)
- static void **clear** (origin_type *po, const iterator &it, const iterator &ite)
- static void **do_clear** (this_type &v)
- static reference **access** (porigin_type, const const_iterator &it, const const_iterator &, size_type i)

template<typename VECT, size_t DIM, size_t NDIM> struct gmm::linalg_traits< sparseSubdimensionReference< const VECT, DIM, NDIM > >

1.7.29 gmm::linalg_traits< sparseSubdimensionReference< VECT, DIM, NDIM > > Struct Template Reference

Public Types

- typedef sparseSubdimensionReference< VECT, DIM, NDIM > **this_type**

- typedef this_type::origin_type **origin_type**
- typedef this_type::porigin_type **porigin_type**
- typedef which_reference< VECT * >::is_reference **is_reference**
- typedef abstract_vector **linalg_type**
- typedef VECT::value_type **value_type**
- typedef this_type::reference **reference**
- typedef this_type::iterator **iterator**
- typedef this_type::const_iterator **const_iterator**
- typedef abstract_sparse **storage_type**
- typedef linalg_false **index_sorted**

Static Public Member Functions

- static size_type **size** (const this_type &v)
- static iterator **begin** (this_type &v)
- static iterator **end** (this_type &v)
- static porigin_type **origin** (this_type &v)
- static void **clear** (origin_type *po, const iterator &it, const iterator &ite)
- static void **do_clear** (this_type &v)
- static reference **access** (origin_type *, const iterator &it, const iterator &, size_type i)

template<typename VECT, size_t DIM, size_t NDIM> struct gmm::linalg_traits< sparseSubdimensionReference< VECT, DIM, NDIM > >

1.7.30 gmm::ref_elt_vector< T, const indefiniteSparseMatrix< T > > Class Template Reference

Public Types

- typedef indefiniteSparseMatrix< T >::key_type **key_type**
- typedef T **value_type**
- typedef const indefiniteSparseMatrix< T > **vect_type**

Public Member Functions

- **operator T** () const
- **ref_elt_vector** (vect_type *p, const key_type &ll)
- ref_elt_vector & **operator=** (T v)
- bool **operator==** (T v) const
- bool **operator!=** (T v) const
- ref_elt_vector & **operator+=** (T v)
- ref_elt_vector & **operator-=** (T v)
- ref_elt_vector & **operator/=** (T v)
- ref_elt_vector & **operator***= (T v)
- ref_elt_vector & **operator=** (const ref_elt_vector &re)
- T **operator+** ()
- T **operator-** ()
- T **operator+** (T v)
- T **operator-** (T v)
- T **operator*** (T v)
- T **operator/** (T v)

Protected Attributes

- vect_type ∗ **pm**
- key_type **l**

template<typename T> class gmm::ref_elt_vector< T, const indefiniteSparseMatrix< T > >

1.7.31 gmm::ref_elt_vector< T, indefiniteSparseMatrix< T > > Class Template Reference

Public Types

- typedef indefiniteSparseMatrix< T >::key_type **key_type**
- typedef T **value_type**
- typedef indefiniteSparseMatrix< T > **vect_type**

Public Member Functions

- **operator T** () const
- **ref_elt_vector** (vect_type ∗p, const key_type &ll)
- ref_elt_vector & **operator=** (T v)
- bool **operator==** (T v) const
- bool **operator!=** (T v) const
- ref_elt_vector & **operator+=** (T v)
- ref_elt_vector & **operator-=** (T v)
- ref_elt_vector & **operator/=** (T v)
- ref_elt_vector & **operator**∗**=** (T v)

- ref_elt_vector & **operator=** (const ref_elt_vector &re)
- T **operator+** ()
- T **operator-** ()
- T **operator+** (T v)
- T **operator-** (T v)
- T **operator*** (T v)
- T **operator/** (T v)

Protected Attributes

- vect_type * **pm**
- key_type **l**

template<typename T> class gmm::ref_elt_vector< T, indefiniteSparseMatrix< T > >

1.7.32 gmm::ref_elt_vector< T, indefiniteVector< T > > Class Template Reference

Public Types

- typedef indefiniteVector< T >::key_type **key_type**
- typedef T **value_type**
- typedef indefiniteVector< T > **vect_type**

Public Member Functions

- **operator T** () const

- **ref_elt_vector** (vect_type *p, const key_type &ll)
- ref_elt_vector & **operator=** (T v)
- bool **operator==** (T v) const
- bool **operator!=** (T v) const
- ref_elt_vector & **operator+=** (T v)
- ref_elt_vector & **operator-=** (T v)
- ref_elt_vector & **operator/=** (T v)
- ref_elt_vector & **operator***= (T v)
- ref_elt_vector & **operator=** (const ref_elt_vector &re)
- T **operator+** ()
- T **operator-** ()
- T **operator+** (T v)
- T **operator-** (T v)
- T **operator*** (T v)
- T **operator/** (T v)

Protected Attributes

- vect_type * **pm**
- key_type **l**

template<**typename T**> **class gmm::ref_elt_vector**< **T, indefiniteVector**< **T** > >

1.7.33 linalg::abstractMatrix< T > Class Template Reference

Public Types

- typedef std::vector< T > **base_class**
- typedef base_class::value_type **value_type**
- typedef base_class::pointer **pointer**
- typedef base_class::const_pointer **const_pointer**
- typedef base_class::reference **reference**
- typedef base_class::const_reference **const_reference**
- typedef base_class::iterator **iterator**
- typedef base_class::const_iterator **const_iterator**
- typedef base_class::const_reverse_iterator **const_reverse_iterator**
- typedef base_class::reverse_iterator **reverse_iterator**
- typedef base_class::size_type **size_type**
- typedef base_class::difference_type **difference_type**
- typedef base_class::allocator_type **allocator_type**

Public Member Functions

- T & **operator()** (const size_t row, const size_t col)
- const T & **operator()** (const size_t row, const size_t col) const
- void **resize** (const size_t NRow, const size_t NCol)
- size_t **nrows** (void) const
- size_t **ncols** (void) const
- bool **writeBinary** (abstractCommHandle *fp) const throw ()
- bool **readBinary** (abstractCommHandle *fp) throw ()

- abstractMatrix & **operator=** (const abstractMatrix &other)
- **abstractMatrix** (const abstractMatrix &other)
- **abstractMatrix** (const size_t NRow, const size_t NCol, const T &t=T())

template<class T> class linalg::abstractMatrix< T >

1.7.34 linalg::linear_interpolator< U, NDIM > Class Template Reference

N-dimensional linear interpolation of values of arbitrary rank.

- uses minimalist N-dimensional matrix semantics as (<linear container>, <shape ntuple>);
- mesh corresponding to data instantiation is specified as ntuple_interval;
- rank of value-type for N-dimensional matrix is arbitrary, but is a fixed-allocation type such as a scalar or ntuple<R,NDIM>.

Public Types

- enum **interp_kind** { **NEAREST** = 0, **NEAREST_PERIODIC**, **LINEAR**, **LINEAR_PERIODIC** }
- typedef gmm::linalg_traits< U >::value_type **value_type**
- typedef tensor_traits< value_type >::scalar_type **S**
- typedef numberTraits< S >::magnitudeType **R**

Public Member Functions

- value_type **operator()** (const ntuple< R, NDIM > &x_i) const throw (std::runtime_error)
- void **apply** (const ntuple< R, NDIM > &x_i, value_type &y_i) const throw (std::runtime_error)
- template<class U1 , class U2 >
  void apply (const U1 &x_i, U2 &y_i) const throw (std::runtime_error)
  interpolation over N-dimensional matrix of co-ordinates.

- **linear_interpolator** (const ntuple_interval< R, NDIM > &domain, const U &y, const ntuple< size_t, NDIM > &shape, interp_kind e) throw (std::runtime_error)
- **linear_interpolator** (const ntuple_interval< R, NDIM > &domain, U &y, const ntuple< size_t, NDIM > &shape, interp_kind e, bool transfer_ownership) throw (std::runtime_error)

1.7.34.1 Detailed Description

template<class U, size_t NDIM> class linalg::linear_interpolator< U, NDIM >

N-dimensional linear interpolation of values of arbitrary rank.

- uses minimalist N-dimensional matrix semantics as (<linear container>, <shape ntuple>);
- mesh corresponding to data instantiation is specified as ntuple_interval;
- rank of value-type for N-dimensional matrix is arbitrary, but is a fixed-allocation type such as a scalar or ntuple<R,NDIM>.

1.7.34.2 Member Function Documentation

1.7.34.3 template<class U , size_t NDIM> template<class U1 , class U2 > void linalg::linear_interpolator< U, NDIM >::apply (const U1 & *x_i*, U2 & *y_i*) const throw (std::runtime_error)

interpolation over N-dimensional matrix of co-ordinates.

Template Parameters

U1 a linear container with value_type compatible with ntuple<R,NDIM>

U2 a linear container with value_type compatible with linear_interpolator<U, NDIM>::value_type

Parameters

`[in]` ***x_i*** coordinate values

`[out]` ***y_i*** interpolated data values

1.7.35 linalg::nonresidentObjectHandler Class Reference

Classes

- struct **object_info**
- struct **sigev_value_struct**
- struct task_data
- class task_data_list

Public Types

- enum **object_state** {

  NONE = -1, **IN** = 0, **OUT** = 1, **EMPTY**,

  N_STATES = 2 }
- enum **SYNCH_MODE** { **SYNCH**, **ASYNCH**, **ASYNCH_BY_LIST** }
- typedef unsigned long **key**

Public Member Functions

- SYNCH_MODE **synch_mode** (SYNCH_MODE e)
- template<class U >

  bool swap (const key &k, U &u, object_state e) throw ()

  blocking, public method (state is well-defined after method, ensures no pending swaps):
- bool **release** (const key &k, bool force=false) throw ()
- bool **attach** (const key &k) throw ()
- void **forget** (const key &k) throw ()
- template<class U >

  bool submitSwap (const key &k, U &u, object_state e) throw ()

  non-blocking method
- template<class U >

  bool submitSwap (const task_data_list &tasks, object_state e) throw ()
- bool release (const task_data_list &tasks, bool force=false) throw ()
- bool **attach** (const task_data_list &tasks) throw ()
- void sync (const key &k) throw ()

blocking, public method

- object_info & **info** (const key &k) const throw ()
- double **swapEfficiency** (object_state e) const throw ()
- void **copy** (const nonresidentObjectHandler &other)
- nonresidentObjectHandler & **operator=** (const nonresidentObjectHandler &other)

Static Public Member Functions

- static SYNCH_MODE **synch_mode** (void)

1.7.35.1 Member Function Documentation

1.7.35.2 bool linalg::nonresidentObjectHandler::release (const task_data_list & *tasks*, bool *force* = `false`) throw ()

list of objects: corresponding attach and release methods:

1.7.35.3 template<class U > bool linalg::nonresidentObjectHandler::submitSwap (const task_data_list & *tasks*, object_state *e*) throw ()

List of objects (sequential media offset only). (task contents are copied to internal list, which is de-allocated at completion of "sync_") Overall key, object-mutex lock, and reference-count for entire action will be that of first object in list, though _-all_ object reference-counts, and states will be updated appropriately. Even though "tasks" is "const", internal "pinfo" field of task_data objects _will_ be modified

by this call. This is a compromise which allows object-info specifics to be dealt with separately from all allocation/de-allocation specifics (the former by the submit method, the latter by "attach_aio_buffer" and "release_aio_buffer" methods).

Due to the first-object-in-list lock association, _all_ threads must use only _one_ form of submitSwap on the same set of objects (prior to completion of the associated sync(<key of first object in list>)).

1.7.35.4 template<class U > bool linalg::nonresidentObjectHandler::submitSwap (const key & *k*, U & *u*, object_state *e*) throw ()

non-blocking method

non-blocking, public method: return value indicates change in state (possibly intermediate through buffer -- state is not valid until associated "sync" completes). Protocols:

1. object_info reference count is associated with "sync" event (possibly implicit), therefore it is not modified by this method.

2. multiple application of submitSwap to same object without intervening sync (or implicit sync via "swap") is an error).

1.7.35.5 template<class U > bool linalg::nonresidentObjectHandler::swap (const key & *k*, U & *u*, object_state *e*) throw ()

blocking, public method (state is well-defined after method, ensures no pending swaps):

blocking, public method: return value indicates change in state.

1.7.35.6 void linalg::nonresidentObjectHandler::sync (const key & *k*) throw ()

blocking, public method

Blocks till completing of any pending asynchronous I/O operations. Adjusts state of info appropriately. Deallocates aio-control-block and transfer buffers

1.7.36 linalg::nonresidentObjectHandler::task_data Struct Reference

Public Member Functions

- **task_data** (const key &k_, void ∗pobject_, void ∗pinfo_)
- **task_data** (const key &k_, void ∗pobject_)

Public Attributes

- key **k**
- void ∗ **pobject**
- void ∗ **pinfo**

1.7.37 linalg::nonresidentObjectHandler::task_data_list Class Reference

Public Member Functions

- const task_data & **operator[ ]** (size_t n) const
- task_data & **operator[ ]** (size_t n)
- size_t **size** (void) const

- bool **empty** (void) const
- task_data * **begin** (void)
- const task_data * **begin** (void) const
- task_data * **end** (void)
- const task_data * **end** (void) const
- void **copy** (const task_data_list &other)
- task_data_list & **operator=** (const task_data_list &other)
- **task_data_list** (size_t n)
- **task_data_list** (const task_data_list &other)

1.7.38 linalg::ntuple< T, DIM > Class Template Reference

Public Types

- typedef numberTraits< T >::magnitudeType **R**
- typedef ntuple **this_type**
- typedef T **value_type**
- typedef T * **iterator**
- typedef const T * **const_iterator**

Public Member Functions

- T & **operator[ ]** (size_t n)
- const T & **operator[ ]** (size_t n) const
- T * **begin** ()
- const T * **begin** () const

- T ∗ **end** ()
- const T ∗ **end** () const
- size_t **size** (void) const throw ()
- void **resize** (size_t) throw ()
- R **dist** (const ntuple &other) const
- T **operator**∗ (const ntuple &other) const
- R **abs** (void) const
- R **absSqr** (void) const
- size_t **numberNonzeroElements** (const R &eps=epsilon< R >()) const
- size_t **nnz** (const R &eps=epsilon< R >()) const
- size_t **mask_nz** (void) const
- void **normalize** (void)
- void **invert** (bool spherical=false)
- void **clear** (void)
- bool **operator==** (const ntuple< T, DIM > &other) const
- bool **operator!=** (const ntuple< T, DIM > &other) const
- ntuple< T, DIM > & **operator+** (void)
- ntuple< T, DIM > **operator-** (void) const
- ntuple< T, DIM > **operator+** (const ntuple< T, DIM > &other) const
- ntuple< T, DIM > **operator-** (const ntuple< T, DIM > &other) const
- ntuple< T, DIM > **operator**∗ (const T &r) const
- ntuple< T, DIM > **operator/** (const T &r) const
- ntuple< T, DIM > **mod** (const T &r) const
- ntuple< T, DIM > & **operator+=** (const ntuple< T, DIM > &other)

- ntuple< T, DIM > & **operator-=** (const ntuple< T, DIM > &other)
- ntuple< T, DIM > & **operator*=** (const T &r)
- ntuple< T, DIM > & **operator/=** (const T &r)
- ntuple< T, DIM > & **operator=** (const T &r)
- ntuple< T, DIM > & **mod_assign** (const T &r)
- T **dot** (const ntuple< T, DIM > &other) const
- template<class S >
  T **dot** (const ntuple< S, DIM > &other) const
- ntuple< T, DIM > **cross** (const ntuple< T, DIM > &other) const
- template<class S >
  ntuple< T, DIM > **cross** (const ntuple< S, DIM > &other) const
- bool **writeBinary** (abstractCommHandle *fp) const
- bool **readBinary** (abstractCommHandle *fp)
- size_t **binarySize** (void)
- void **write** (std::ostream &os) const
- void **read** (std::istream &is)
- void **debug_print** (void) const
- **ntuple** (bool clear=true)
- **ntuple** (size_t DIM_)
- **ntuple** (const T &e1)
- **ntuple** (const T &e1, const T &e2)
- **ntuple** (const T &e1, const T &e2, const T &e3)
- template<class T1 >
  ntuple (const T1 &e1, const T1 &e2, const T1 &e3)

Static Public Member Functions

- static ntuple< T, DIM > **unit** (void)
- static ntuple< T, DIM > **filled** (const T &t)
- static void **sort** (ntuple< T, DIM > &p) throw ()
- static void **apply** (T(∗func)(const T &), const ntuple< T, DIM > &src, ntuple< T, DIM > &dest) throw ()

template<class T, size_t DIM = 3> class linalg::ntuple< T, DIM >

1.7.39 linalg::ntuple_interval< T, DIM > Class Template Reference

Public Member Functions

- const ntuple< T, DIM > & **start** (void) const
- ntuple< T, DIM > & **start** (void)
- const ntuple< T, DIM > & **end** (void) const
- ntuple< T, DIM > & **end** (void)
- const T & **left_epsilon** (void) const
- const T & **right_epsilon** (void) const
- bool **interior** (const ntuple< T, DIM > &p) const
- bool **boundary** (const ntuple< T, DIM > &p) const
- bool **exterior** (const ntuple< T, DIM > &p) const
- bool **closure** (const ntuple< T, DIM > &p) const
- bool **left_closure** (const ntuple< T, DIM > &p) const
- bool **right_closure** (const ntuple< T, DIM > &p) const

- bool **operator==** (const ntuple_interval< T, DIM > &other) const
- bool **operator!=** (const ntuple_interval< T, DIM > &other) const
- bool **writeBinary** (abstractCommHandle *fp) const
- bool **readBinary** (abstractCommHandle *fp)
- size_t **binarySize** (void) const
- void **write** (std::ostream &os, const ntuple< T, DIM > *p=NULL) const
- ntuple< T, DIM > scale (void) const

  Calculate scale tuple corresponding to this ntuple-interval (i.e. as end() - start()).

- **ntuple_interval** (const ntuple< T, DIM > &start, const ntuple< T, DIM > &end, const T &left_eps, const T &right_eps)
- ntuple_interval (const ntuple< T, DIM > &start, const ntuple< T, DIM > &end, const T &eps=zero< T >())

  this signature assumes left_eps == right_eps:

Static Public Member Functions

- template<class U >

  static ntuple_interval extent (const U &u, const T &left_eps, const T &right_-eps)

  Calculate extent of a container of ntuple<T,DIM> as ntuple_interval<T,DIM> Assumes typename U::value_type is ntuple<T,DIM>

- template<class U >

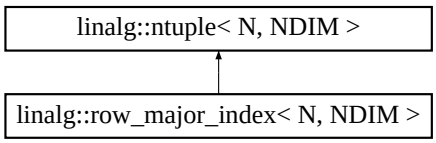
  static ntuple_interval extent (const U &u, const T &eps=zero< T >())

  Calculate extent of a container of ntuple<T,DIM> as ntuple_interval<T,DIM> Assumes typename U::value_type is ntuple<T,DIM> (this signature assumes left_epsilon == right_epsilon).

template<class T, size_t DIM> class linalg::ntuple_interval< T, DIM >

1.7.40 linalg::row_major_index< N, NDIM > Class Template Reference

Inheritance diagram for linalg::row_major_index< N, NDIM >:



Public Types

- typedef ntuple< N, NDIM > **base_class**
- typedef base_class::value_type **value_type**
- typedef ntuple< size_t, NDIM > **shape_type**

Public Member Functions

- row_major_index< N, NDIM > **operator+** (const row_major_index &other) const
- row_major_index< N, NDIM > **operator-** (const row_major_index &other) const
- row_major_index< N, NDIM > & **operator+=** (const row_major_index &other)
- row_major_index< N, NDIM > & **operator-=** (const row_major_index &other)
- row_major_index< N, NDIM > & **mod_assign** (const shape_type &shape)

- row_major_index< N, NDIM > & **operator=** (const row_major_index &other)
- template<class N1 >
  row_major_index< N, NDIM > & **operator=** (const ntuple< N1, NDIM > &other)
- size_t **inverse** (const shape_type &shape, value_type shift=0) const
- bool **writeBinary** (commUtil::abstractCommHandle *fp) const throw (std::string)
- bool **readBinary** (commUtil::abstractCommHandle *fp) throw (std::string)
- size_t **binarySize** (void) const throw (std::string)
- bool **in_domain** (const shape_type &shape, value_type shift=0) const
- **row_major_index** (value_type shift=0, bool clear=true)
- **row_major_index** (size_t n_linear, const shape_type &shape, value_type shift=0)
- **row_major_index** (const row_major_index &other)
- template<class N1 >
  row_major_index (const ntuple< N1, NDIM > &other)

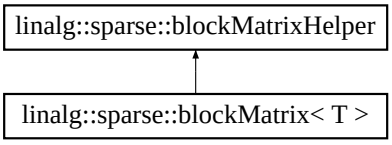
Static Public Member Functions

- static bool **wrap** (const row_major_index &x1, const row_major_index &x2, const shape_type &shape, row_major_index &wrap_dim)
- template<class R >
  static row_major_index< N, NDIM > **ND_bin** (const ntuple< R, NDIM > &p, const ntuple_interval< R, NDIM > &domain_cube, const shape_type &shape, value_type shift=0) throw (std::string)

template<class N, size_t NDIM> class linalg::row_major_index< N, NDIM >

1.7.41 linalg::sparse::blockMatrix< T > Class Template Reference

Inheritance diagram for linalg::sparse::blockMatrix< T >:



Public Types

- typedef numberTraits< T >::magnitudeType **R**

Public Member Functions

- bool **allocated** (void) const
- bool **writeBinary** (abstractCommHandle *fp) const
- bool **readBinary** (abstractCommHandle *fp)
- size_t **binarySize** (void) const
- void **write** (std::ostream &os) const
- void **writeSignature** (std::ostream &os, bool extendedInfo=false) const
- std::string **md5sum** (void) const
- void **debug_print** (void) const
- void **blockSize** (size_t &NRows, size_t &NCols) const throw ()
- bool **pseudoInverse** (blockMatrix< T > &dest) const throw ()

- T **trace** (void) const throw ()
- T **mask_trace** (const std::vector< long > &vnRowMask, const std::vector< long > &vnColMask) const throw ()
- void **clear** (void)
- void **clean** (const R &eps_=epsilon< R >(), bool deallocate=true)
- blockMatrix< T > & **operator+=** (const blockMatrix< T > &other)
- blockMatrix< T > & **operator-=** (const blockMatrix< T > &other)
- blockMatrix< T > **operator*** (const blockMatrix< T > &other) const
- blockMatrix< T > & **operator*=** (const T &scale)
- std::vector< long > & **vnRows** (void)
- const std::vector< long > & **vnRows** (void) const
- std::vector< long > & **vnCols** (void)
- const std::vector< long > & **vnCols** (void) const
- size_t **NBlocks** (void) const
- const index2DFunctor & **indexer** (void) const
- std::vector< gmm::dense_matrix< T > > & **vBlocks** (void)
- const std::vector< gmm::dense_matrix< T > > & **vBlocks** (void) const
- blockMatrix< T > & **operator=** (const blockMatrix< T > &other)
- void **assignFromFields** (std::vector< gmm::dense_matrix< T > > &vBlocks, std::vector< long > &vnRows, std::vector< long > &vnCols, const index2DFunctor &indexer, bool ownsData=false)
- **blockMatrix** (const blockMatrix< T > &other)
- **blockMatrix** (const blockMatrix< T > &other, bool)
- **blockMatrix** (std::vector< gmm::dense_matrix< T > > &vBlocks, std::vector< long > &vnRows, std::vector< long > &vnCols, const index2DFunctor &indexer, bool ownsData=false)

Static Public Member Functions

- static void **mult** (const blockMatrix< T > &src1, const blockMatrix< T > &src2, blockMatrix< T > &dest, bool pseudo=false)
- static void **mult** (const blockMatrix< T > &src1, const blockVector< T > &src2, blockVector< T > &dest, bool pseudo=false)
- static blockMatrix< T > **identity** (const size_t nrows, const size_t ncols, const std::vector< long > &vnRows, const std::vector< long > &vnCols, const index2DFunctor &indexer)
- static void **identity** (const size_t nrows, const size_t ncols, const std::vector< long > &vnRows, const std::vector< long > &vnCols, const index2DFunctor &indexer, blockMatrix< T > &bI, bool reinitialize=false)
- static blockMatrix< T > **zero** (const size_t nrows, const size_t ncols, const std::vector< long > &vnRows, const std::vector< long > &vnCols, const index2DFunctor &indexer)
- static void **zero** (const size_t nrows, const size_t ncols, const std::vector< long > &vnRows, const std::vector< long > &vnCols, const index2DFunctor &indexer, blockMatrix< T > &bZ, bool reinitialize=false)
- static void **scrambleCopy** (const blockMatrix< T > &src, const std::vector< long > &vnDestRows, const std::vector< long > &vnDestCols, blockMatrix< T > &dest) throw ()
- template<class M >

  static void **flattenCopy** (const blockMatrix< T > &src, M &dest, bool strict_l=true) throw ()
- template<class M >

  static void **flattenCopy** (const blockMatrix< T > &src, const M &dest, bool strict_l=true) throw ()

- template<class M >

  static void **unflattenCopy** (const M &src, blockMatrix< T > &dest, bool strict_l=true) throw ()
- static void **getRow** (const blockMatrix< T > &src, size_t nblockRow, size_t nrow, blockVector< T > &dest) throw ()
- static void **getCol** (const blockMatrix< T > &src, size_t nblockCol, size_t ncol, blockVector< T > &dest) throw ()
- static void **setRow** (const blockVector< T > &src, size_t nblockRow, size_t nrow, blockMatrix< T > &dest) throw ()
- static void **setCol** (const blockVector< T > &src, size_t nblockCol, size_t ncol, blockMatrix< T > &dest) throw ()

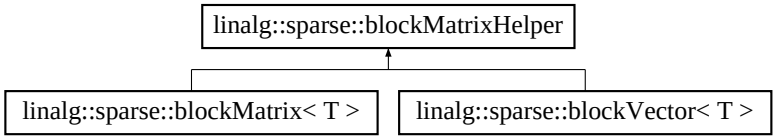
Protected Member Functions

- **blockMatrix** (const std::vector< long > &vnRows, const std::vector< long > &vnCols, const index2DFunctor &indexer)
- bool **LUFactor** (blockMatrix< T > &LU, T &rowPermutationParity) const throw ()
- void **LUInverseColumn** (blockMatrix< T > &aInv, const size_t &nnInvCol) const throw ()

template<class T> class linalg::sparse::blockMatrix< T >

1.7.42 linalg::sparse::blockMatrixHelper Class Reference

Inheritance diagram for linalg::sparse::blockMatrixHelper:

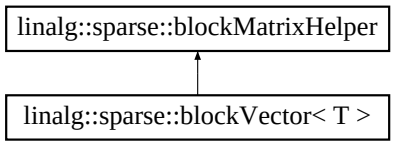


Static Public Member Functions

- static void **setStatusString** (const char ∗msg)
- static const std::string & **getStatusString** (void)
- static void **mergeIndexVectors** (const std::vector< long > &vnSrc1, const std::vector< long > &vnSrc2, std::vector< long > &vnDest)
- static size_t **flattenedIndex** (size_t nblock, size_t nelt, size_t N_nontrivial_-blocks, size_t block_size, bool strict_l=true) throw ()
- static void **inverseFlattenedIndex** (size_t n, size_t N_nontrivial_blocks, size_t block_size, size_t &nblock, size_t &nelt, bool strict_l=true) throw ()

1.7.43 linalg::sparse::blockVector< T > Class Template Reference

Inheritance diagram for linalg::sparse::blockVector< T >:



Public Types

- typedef numberTraits< T >::magnitudeType **R**

Public Member Functions

- bool **allocated** (void) const
- bool **writeBinary** (abstractCommHandle ∗fp) const
- bool **readBinary** (abstractCommHandle ∗fp)
- size_t **binarySize** (void) const
- void **write** (std::ostream &os) const
- void **writeSignature** (std::ostream &os, bool extendedInfo=false) const
- std::string **md5sum** (void) const
- void **debug_print** (void) const
- void **blockSize** (size_t &N) const throw ()
- size_t **blockSize** (void) const throw ()
- void **clear** (void)
- void **clean** (const R &eps_=epsilon< R >(), bool deallocate=true)
- blockVector< T > & **operator+=** (const blockVector< T > &other)
- blockVector< T > & **operator-=** (const blockVector< T > &other)
- blockVector< T > & **operator∗=** (const T &scale)
- std::vector< long > & **vnBlocks** (void)
- const std::vector< long > & **vnBlocks** (void) const
- size_t **NBlocks** (void) const
- std::vector< std::vector< T > > & **vBlocks** (void)
- const std::vector< std::vector< T > > & **vBlocks** (void) const
- blockVector< T > & **operator=** (const blockVector< T > &other)
- void **assignFromFields** (std::vector< std::vector< T > > &vBlocks, std::vector< long > &vnBlocks, bool ownsData=false)

- **blockVector** (const blockVector< T > &other)
- **blockVector** (const blockVector< T > &other, bool)
- **blockVector** (std::vector< std::vector< T > > &vBlocks, std::vector< long > &vnBlocks, bool ownsData=false)

Static Public Member Functions

- static blockVector< T > **zero** (const size_t block_size, const std::vector< long > &vnBlocks, size_t N_blocks)
- static void **zero** (const size_t block_size, const std::vector< long > &vnBlocks, size_t N_blocks, blockVector< T > &bZ, bool reinitialize=false)
- static void **scrambleCopy** (const blockVector< T > &src, const std::vector< long > &vnDestBlocks, blockVector< T > &dest) throw ()
- template<class V >
  static void **flattenCopy** (const blockVector< T > &src, V &dest, bool strict_l=true) throw ()
- template<class V >
  static void **unflattenCopy** (const V &src, blockVector< T > &dest, bool strict_l=true) throw ()

Protected Member Functions

- **blockVector** (const std::vector< long > &vnRows, size_t N_blocks)

template<class T> class linalg::sparse::blockVector< T >

1.7.44 linalg::sparse::compressed1DIndex Class Reference

Public Types

- typedef size_t **size_type**

Public Member Functions

- bool **indexExists** (const size_t &n) const
- size_type **compressedIndex** (size_type n) const throw ()
- size_type **index** (size_type compressedIndex_) const throw ()
- size_type **compressedSize** (void) const
- size_type **size** (void) const
- const std::vector< size_type > & **vnReverse** (void) const
- template<class T >
  T & **deref** (blockVector< T > &v, size_type nBlock, size_type n) const throw ()
- template<class T >
  T & **deref** (std::vector< T > &vBlock, size_type n) const throw ()
- template<class T >
  const T & **deref** (const blockVector< T > &v, size_type nblock, size_type n) const throw ()
- template<class T >
  const T & **deref** (const std::vector< T > &vBlock, size_type n) const throw ()
- compressed1DIndex & **operator=** (const compressed1DIndex &other)
- bool **writeBinary** (abstractCommHandle *fp) const throw ()
- bool **readBinary** (abstractCommHandle *fp) throw ()

- **compressed1DIndex** (const compressed1DIndex &other)
- **compressed1DIndex** (const std::vector< size_type > &vnReverse) throw ()

- void **init** (const std::vector< size_type > &vnReverse) throw ()
- **compressed1DIndex** (const std::vector< long > &vnReverse) throw ()
- void **init** (const std::vector< long > &vnReverse) throw ()

Static Public Attributes

- static const size_t **not_an_index**

1.7.45 linalg::sparse::compressed2DIndex Class Reference

Public Types

- typedef compressed1DIndex::size_type **size_type**

Public Member Functions

- bool **indexExists** (const size_t &row, const size_t &col) const
- bool **rowExists** (const size_t &row) const
- bool **colExists** (const size_t &col) const
- size_type **compressedRow** (size_type row) const throw ()
- size_type **compressedCol** (size_type col) const throw ()
- size_type **row** (size_type compressedRow) const throw ()
- size_type **col** (size_type compressedCol) const throw ()
- size_type **n_compressedRows** (void) const

- size_type **n_compressedCols** (void) const
- size_type **nrows** (void) const
- size_type **ncols** (void) const
- const std::vector< size_type > & **vnReverseRows** (void) const
- const std::vector< size_type > & **vnReverseCols** (void) const
- const compressed1DIndex & **rowCompressor** (void) const
- const compressed1DIndex & **columnCompressor** (void) const
- template<class T >
  T & **deref** (blockMatrix< T > &a, size_type blockRow, size_type blockCol, size_type row, size_type col) const throw ()
- template<class T >
  T & **deref** (typename gmm::dense_matrix< T > &aBlock, size_type row, size_type col) const throw ()
- template<class T >
  const T & **deref** (const blockMatrix< T > &a, size_type blockRow, size_type blockCol, size_type row, size_type col) const throw ()
- template<class T >
  const T & **deref** (const typename gmm::dense_matrix< T > &aBlock, size_type row, size_type col) const throw ()
- compressed2DIndex & **operator=** (const compressed2DIndex &other)
- bool **writeBinary** (abstractCommHandle *fp) const throw ()
- bool **readBinary** (abstractCommHandle *fp) throw ()
- **compressed2DIndex** (const compressed2DIndex &other)
- **compressed2DIndex** (const std::vector< size_type > &vnReverseRows, const std::vector< size_type > &vnReverseColumns) throw ()

- void **init** (const std::vector< size_type > &vnReverseRows, const std::vector< size_type > &vnReverseColumns) throw ()
- **compressed2DIndex** (const std::vector< long > &vnReverseRows, const std::vector< long > &vnReverseColumns) throw ()
- void **init** (const std::vector< long > &vnReverseRows, const std::vector< long > &vnReverseColumns) throw ()

1.7.46 linalg::sparse::indefiniteSparseMatrix< T > Class Template Reference

Public Types

- typedef indefiniteSparseMatrix< T > **this_type**
- typedef _STL_EXT_NAMESPACE_::hash_map< nDimensionalKey< unsigned long, 2 >, T > **base_class**
- typedef nDimensionalKey< unsigned long, 2 > **key_type**
- typedef T **value_type**
- typedef gmm::ref_elt_vector< T, this_type > **reference**
- typedef numberTraits< T >::POD_VALUE_RETURN **const_reference**
- typedef base_class::size_type **size_type**

Public Member Functions

- const_reference **operator()** (size_type l, size_type c) const
- reference **operator()** (size_type l, size_type c)
- void **w** (const key_type &key, const T &e)

- const_reference **r** (const key_type &key) const
- size_type **nb_stored** (void) const
- const size_type **nrows** (void) const
- const size_type **ncols** (void) const
- void **restrictSize** (const size_type &newNRows, const size_type &newNCols) const
- void **unrestrictSize** (void) const
- const size_type **actual_nrows** (void) const
- const size_type **actual_ncols** (void) const
- const size_type **actual_size** (size_t dim) const
- void **nnzByIndex** (std::map< long, long > &mRowN, std::map< long, long > &mColN) const
- void **swap** (this_type &v)
- void **resize** (size_type, size_type)
- void **reshape** (size_type, size_type)
- void **fill** (const T &b=gmm::number_traits< T >::zero())
- template<typename M >
  void **toCompressedFormat** (M &dest, std::vector< long > &vRowN, std::vector< long > &vColN) const throw ()
- template<typename M >
  void **toCommonCompressedFormat** (M &dest, std::vector< long > &vRowN, std::vector< long > &vColN) const throw ()
- template<typename M >
  void **fromCompressedFormat** (const M &src, const std::vector< long > &vRowN, const std::vector< long > &vColN) throw ()

- template<typename M >
  void **toDenseFormat** (M &dest) const throw ()
- template<typename M >
  void **fromDenseFormat** (const M &src) throw ()
- bool **writeBinary** (abstractCommHandle *fp) const
- bool **readBinary** (abstractCommHandle *fp)
- void **init** (size_type l, size_type c)
- **indefiniteSparseMatrix** (size_type l, size_type c)

Static Public Member Functions

- static void **transpose** (const this_type &src, this_type &dest) throw ()
- static void **hermitianConjugate** (const this_type &src, this_type &dest) throw ()

Protected Attributes

- size_type **nRestrictedRows**
- size_type **nRestrictedColumns**

template<typename T> class linalg::sparse::indefiniteSparseMatrix< T >

1.7.47 linalg::sparse::indefiniteVector< T > Class Template Reference

Public Types

- typedef indefiniteVector< T > **this_type**
- typedef nDimensionalKey< unsigned long, 1 > **key_type**

- typedef T **value_type**
- typedef _STL_EXT_NAMESPACE_::hash_map< key_type, T > **base_class**
- typedef base_class::size_type **size_type**
- typedef base_class::iterator **base_iterator**
- typedef base_class::const_iterator **base_const_iterator**
- typedef sparseNDimensional_iterator< this_type, 1, 1 > **iterator**
- typedef sparseNDimensional_iterator< const this_type, 1, 1 > **const_iterator**
- typedef std::reverse_iterator< const_iterator > **const_reverse_iterator**
- typedef std::reverse_iterator< iterator > **reverse_iterator**

Public Member Functions

- iterator **begin** (void)
- const_iterator **begin** (void) const
- iterator **end** (void)
- const_iterator **end** (void) const
- reverse_iterator **rbegin** (void)
- const_reverse_iterator **rbegin** (void) const
- reverse_iterator **rend** (void)
- const_reverse_iterator **rend** (void) const
- void **clean** (const T &eps)
- void **resize** (size_type)
- gmm::number_traits< T >::POD_VALUE_RETURN **operator[ ]** (size_type c) const

- gmm::ref_elt_vector< T, indefiniteVector< T > > **operator[ ]** (size_type c)

- void **w** (const key_type &key, const T &e)
- gmm::number_traits< T >::POD_VALUE_RETURN **r** (const key_type &key) const
- size_type **nb_stored** (void) const
- void **nnzByIndex** (std::map< long, long > &mN) const
- const size_type **size** (void) const
- void **restrictSize** (const size_type &newRestriction) const
- void **unrestrictSize** (void) const
- const size_type **actualSize** (void) const
- void **swap** (indefiniteVector< T > &v)
- template<typename V >
  void **toCompressedFormat** (V &dest, std::vector< long > &vN) const throw ()
- template<typename V >
  void **toCommonCompressedFormat** (V &dest, std::vector< long > &vN) const throw ()
- template<typename V >
  void **fromCompressedFormat** (const V &src, const std::vector< long > &vN) throw ()
- template<typename V >
  void **toDenseFormat** (V &dest) const throw ()
- template<typename V >
  void **fromDenseFormat** (const V &src) throw ()

- bool **writeBinary** (abstractCommHandle ∗fp) const
- bool **readBinary** (abstractCommHandle ∗fp)
- void **init** (size_type l)
- **indefiniteVector** (size_type l)

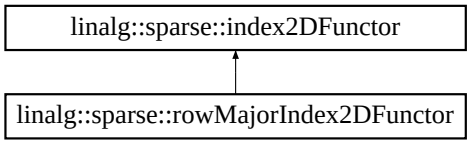
Protected Attributes

- size_type **nRestrictedElements**

template<**typename T**> **class linalg::sparse::indefiniteVector**< **T** >

1.7.48 linalg::sparse::index2DFunctor Class Reference

Inheritance diagram for linalg::sparse::index2DFunctor:



Public Types

- typedef size_t **size_type**
- typedef size_type(∗ **p_index_function**)(const size_type &, const size_type &)
- typedef void(∗ **p_inverse_index_function**)(const size_type &, size_type &, size_type &)

Public Member Functions

- virtual size_type **operator()** (const long &row, const long &col) const =0
- virtual void **inverse** (const size_type &index1D, long &row, long &col) const =0
- virtual size_type **size1D** (void) const
- virtual size_type **nrows** (void) const =0
- virtual size_type **ncols** (void) const =0
- virtual index2DFunctor * **clone** (void) const =0
- virtual bool **writeBinary** (abstractCommHandle *fp) const throw ()
- virtual bool **readBinary** (abstractCommHandle *fp) throw ()
- virtual size_t **binarySize** (void) const throw ()

1.7.49 linalg::sparse::lessThanDim1< ITER > Struct Template Reference

Public Types

- typedef ITER **first_argument_type**
- typedef ITER **second_argument_type**
- typedef bool **result_type**
- typedef ITER::value_type **DREF**

Public Member Functions

- bool **operator()** (const DREF &val1, const DREF &val2) const

template<class ITER> struct linalg::sparse::lessThanDim1< ITER >

1.7.50 linalg::sparse::lessThanDim2< ITER > Struct Template Reference

Public Types

- typedef ITER **first_argument_type**
- typedef ITER **second_argument_type**
- typedef bool **result_type**
- typedef ITER::value_type **DREF**

Public Member Functions

- bool **operator()** (const DREF &val1, const DREF &val2) const

template<class ITER> struct linalg::sparse::lessThanDim2< ITER >

1.7.51 linalg::sparse::maskOtherDim< ITER, NDIM > Struct Template Reference

Public Types

- typedef nDimensionalKey< unsigned long, NDIM > **key_type**
- typedef ITER::value_type **DREF**
- typedef DREF **first_argument_type**
- typedef size_t **second_argument_type**
- typedef key_type **result_type**

Public Member Functions

- key_type **operator()** (const DREF &val, const size_t dim) const

template<class ITER, size_t NDIM> struct linalg::sparse::maskOtherDim< ITER, NDIM >

1.7.52 linalg::sparse::nDimensionalKey< index_type_, N > Class Template Reference

Public Types

- typedef nDimensionalKey< index_type_, N > **this_type**
- typedef index_type_ **index_type**
- typedef unsigned long **size_type**
- typedef ptrdiff_t **difference_type**

Public Member Functions

- **nDimensionalKey** (const this_type &other)
- **nDimensionalKey** (const index_type &x1)
- **nDimensionalKey** (const index_type &x2, const index_type &x1)
- **nDimensionalKey** (const index_type &x3, const index_type &x2, const index_type &x1)
- bool **operator<** (const this_type &other) const
- bool **operator==** (const this_type &other) const
- bool **lessThanSubDim** (const this_type &other, size_type dim=1) const
- bool **equalSubDim** (const this_type &other, size_type dim=1) const

- bool **operator!=** (const this_type &other) const
- const index_type & **index** (const size_type n=1) const
- index_type & **index** (const size_type n=1)
- **operator size_t** (void) const
- void **write** (std::ostream &os) const
- bool **writeBinary** (abstractCommHandle *fp) const
- bool **readBinary** (abstractCommHandle *fp)

Static Public Member Functions

- static this_type **subDimStart** (const this_type &fromKey, const size_type n=1)

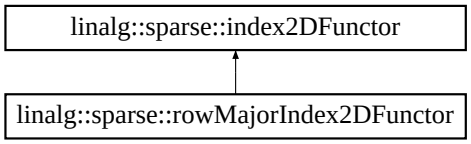
Protected Attributes

- index_type **keyData** [N]

template<typename index_type_, size_t N> class linalg::sparse::nDimensionalKey< index_type_, N >

1.7.53 linalg::sparse::rowMajorIndex2DFunctor Class Reference

Inheritance diagram for linalg::sparse::rowMajorIndex2DFunctor:



Public Member Functions

- virtual size_type **operator()** (const long &row, const long &col) const
- virtual void **inverse** (const size_type &index1D, long &row, long &col) const

- rowMajorIndex2DFunctor & **operator=** (const rowMajorIndex2DFunctor &other)
- virtual size_type **nrows** (void) const
- virtual size_type **ncols** (void) const
- virtual index2DFunctor * **clone** (void) const
- virtual bool **writeBinary** (abstractCommHandle *fp) const throw ()
- virtual bool **readBinary** (abstractCommHandle *fp) throw ()
- virtual size_t **binarySize** (void) const throw ()
- **rowMajorIndex2DFunctor** (const rowMajorIndex2DFunctor &other)
- **rowMajorIndex2DFunctor** (const size_type &nrows, const size_type &ncols)
- void **init** (const size_type &nrows, const size_type &ncols)

Protected Member Functions

- size_type **forwardIndex** (const size_type &row, const size_type &col) const

- void **inverseIndex** (const size_type &index, size_type &row, size_type &col) const

1.7.54 linalg::sparse::sparseNDimensional_iterator< VECT, DIM, NDIM > Class Template Reference

Public Types

- typedef unsigned long **size_type**
- typedef nDimensionalKey< unsigned long, NDIM > **key_type**
- typedef VECT::value_type **value_type**
- typedef gmm::linalg_traits< VECT >::origin_type **origin_type**
- typedef propagate_const2< origin_type *, const origin_type *, VECT >::result_type **porigin_type**
- typedef value_type * **pointer**
- typedef const value_type * **const_pointer**
- typedef propagate_const2< gmm::ref_elt_vector< value_type, origin_type >, const gmm::ref_elt_vector< value_type, const origin_type >, VECT >::result_type **reference**
- typedef ptrdiff_t **difference_type**
- typedef std::bidirectional_iterator_tag **iterator_category**
- typedef sparseNDimensional_iterator< VECT, DIM, NDIM > **this_type**
- typedef _STL_EXT_NAMESPACE_::hash_map< key_type, value_type > **base_class**
- typedef VECT::this_type **basic_origin_type**
- typedef propagate_const2< const basic_origin_type, basic_origin_type, VECT >::result_type **other_const_type**

Public Member Functions

- this_type & **operator++** ()

- this_type & **operator--** ()
- this_type **operator++** (int)
- this_type **operator--** (int)
- this_type & **operator+=** (size_type n)
- this_type & **operator-=** (size_type n)
- this_type **operator+** (size_type n)
- this_type **operator-** (size_type n)
- const unsigned long & **index** (void) const
- const key_type & **key** (void) const
- bool **operator==** (const this_type &other) const
- bool **operator!=** (const this_type &other) const
- bool **operator**< (const this_type &other) const
- difference_type **operator-** (const this_type &other) const
- reference **operator**∗ (void)
- const value_type & **operator**∗ (void) const
- origin_type & **origin** (void)
- const origin_type & **origin** (void) const
- **sparseNDimensional_iterator** (const sparseNDimensional_iterator< VECT, DIM, NDIM > &other)
- **sparseNDimensional_iterator** (const sparseNDimensional_iterator< other_const_type, DIM, NDIM > &other)
- **sparseNDimensional_iterator** (const VECT &origin, const key_type &key)

Protected Attributes

- key_type **key_**
- porigin_type **porigin_**

template<typename VECT, size_t DIM, size_t NDIM> class linalg::sparse::sparseNDimensional_iterator< VECT, DIM, NDIM >

1.7.55 linalg::sparse::sparseSubdimensionReference< VECT, DIM, NDIM > Class Template Reference

Public Types

- typedef sparseSubdimensionReference< VECT, DIM, NDIM > **this_type**
- typedef nDimensionalKey< unsigned long, NDIM > **key_type**
- typedef gmm::linalg_traits< VECT >::value_type **value_type**
- typedef gmm::linalg_traits< VECT >::origin_type **origin_type**
- typedef propagate_const2< origin_type *, const origin_type *, VECT >::result_type **porigin_type**
- typedef propagate_const2< gmm::ref_elt_vector< value_type, origin_type >, const gmm::ref_elt_vector< value_type, const origin_type >, VECT >::result_type **reference**
- typedef numberTraits< value_type >::POD_VALUE_RETURN **const_reference**
- typedef VECT::size_type **size_type**
- typedef origin_type::difference_type **difference_type**
- typedef sparseNDimensional_iterator< origin_type, DIM, NDIM > **iterator**

- typedef sparseNDimensional_iterator< const origin_type, DIM, NDIM > **const_iterator**
- typedef std::reverse_iterator< const_iterator > **const_reverse_iterator**
- typedef std::reverse_iterator< iterator > **reverse_iterator**
- typedef propagate_const2< iterator, const_iterator, VECT >::result_type **native_iterator**

Public Member Functions

- bool **empty** (void) const
- size_type **size** (void) const
- const iterator & **begin** (void)
- const const_iterator & **begin** (void) const
- const iterator & **end** (void)
- const const_iterator & **end** (void) const
- reverse_iterator **rbegin** (void)
- const_reverse_iterator **rbegin** (void) const
- reverse_iterator **rend** (void)
- const_reverse_iterator **rend** (void) const
- reference **front** (void)
- const_reference **front** (void) const
- reference **back** (void)
- const_reference **back** (void) const
- void **pop_front** (void)
- void **w** (size_type c, const value_type &e)
- const value_type & **r** (size_type c) const

- porigin_type **porigin** (void)
- const_reference **operator[ ]** (size_type c) const
- reference **operator[ ]** (size_type c)
- **sparseSubdimensionReference** (const VECT &v, const native_iterator &b, const native_iterator &e)

Protected Attributes

- porigin_type **porigin_**
- native_iterator **begin_**
- native_iterator **end_**

template<typename VECT, size_t DIM, size_t NDIM> class linalg::sparse::sparseSubdimensionReference< VECT, DIM, NDIM >

1.7.56 linalg::tensor_traits< T > Struct Template Reference

Traits class to facilitate implementation of N-dimensional arrays and container objects with either scalar (RANK = 0) or non-scalar (RANK > 0) elements.

Public Types

- typedef T **scalar_type**

Static Public Attributes

- static const size_t **rank** = 0
- static const size_t **N_components** = 1

1.7.56.1 Detailed Description

template<class T> struct linalg::tensor_traits< T >

Traits class to facilitate implementation of N-dimensional arrays and container objects with either scalar (RANK = 0) or non-scalar (RANK > 0) elements.

1.7.57 linalg::tensor_traits< linalg::ntuple< T, NDIM > > Struct Template Reference

Public Types

- typedef T **scalar_type**

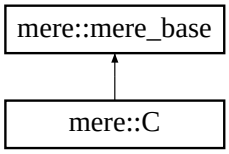
Static Public Attributes

- static const size_t **rank** = 1
- static const size_t **N_components** = NDIM

template<class T, size_t NDIM> struct linalg::tensor_traits< linalg::ntuple< T, NDIM > >

1.7.58 mere::C Class Reference

Inheritance diagram for mere::C:



Public Member Functions

- size_t **thisBits** (void) const
- size_t **thisDigits** (void) const
- R & **real** (void)
- R & **imag** (void)
- const R & **real** (void) const
- const R & **imag** (void) const
- int **compare_abs** (const C &other) const
- int **compare_abs** (const R &other) const
- bool **operator==** (const C &other) const
- bool **operator!=** (const C &other) const
- C **operator+** (void) const
- C **operator+** (const C &other) const
- C & **operator+=** (const C &other)
- C & **operator++** (void)
- C **operator++** (int)
- C **operator-** (void) const
- C **operator-** (const C &other) const
- C & **operator-=** (const C &other)
- C & **operator--** (void)
- C **operator--** (int)
- C **operator*** (const C &other) const
- C & **operator*=** (const C &other)
- C **operator/** (const C &other) const

- C & **operator/=** (const C &other)
- C & **operator+=** (const Z &z)
- C & **operator-=** (const Z &z)
- C & **operator∗=** (const Z &z)
- C & **operator/=** (const Z &z)
- C & **operator=** (const Z &z)
- C & **operator+=** (const R &r)
- C & **operator-=** (const R &r)
- C & **operator∗=** (const R &r)
- C & **operator/=** (const R &r)
- C & **operator=** (long other)
- C **operator+** (long other) const
- C & **operator+=** (long other)
- C **operator-** (long other) const
- C & **operator-=** (long other)
- C **operator∗** (long other) const
- C & **operator∗=** (long other)
- C **operator/** (long other) const
- C & **operator/=** (long other)
- void **read** (std::istream &is, bool allowPrecisionChange=false)
- void **write** (std::ostream &os) const
- bool **readBinary** (abstractCommHandle ∗fp)
- bool **writeBinary** (abstractCommHandle ∗fp) const
- size_t **binarySize** (void) const

- void **debug_print** (void) const
- void **swap** (C &other)
- void **copy** (const C &other)
- C & **operator=** (const C &other)
- C & **operator=** (const R &other)
- C & **operator=** (const char *sC) throw (mere_exception)
- **C** (C &c, bool transferData)
- **C** (long nR, long nI, void *pDataR, void *pDataI, size_t precision)
- **C** (const C &c)
- **C** (const R &r)
- **C** (const R &rR, const R &rI)
- **C** (const char *sC) throw (mere_exception)
- **C** (long nR)
- **C** (int nR)
- **C** (long nR, long nI, size_t precision)

Static Public Member Functions

- static const C & **zero** (void)
- static const C & **zero** (size_t prec)
- static const C & **one** (void)
- static const C & **one** (size_t prec)
- static const C & **one_i** (void)
- static const C & **one_i** (size_t prec)
- static C **random** (void)

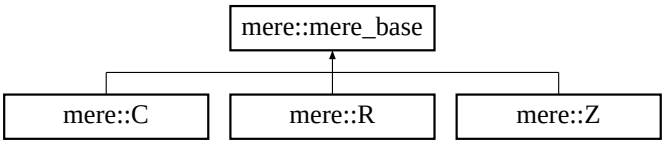
- static C **random** (size_t prec)
- static R **abs** (const C &c)
- static void **abs** (R &, const C &c)
- static R **abs_sqr** (const C &c)
- static void **abs_sqr** (R &, const C &c)
- static R **arg** (const C &c)
- static void **arg** (R &r, const C &c)
- static C **neg** (const C &c)
- static void **neg** (C &val, const C &c)
- static C **inv** (const C &c)
- static void **inv** (C &val, const C &c)
- static C **conj** (const C &c)
- static C **polar** (const R &abs_, const R &arg_)
- static void **polar** (C &val, const R &abs_, const R &arg_)
- static C **sqr** (const C &c)
- static void **sqr** (C &val, const C &c)
- static C **sqrt** (const C &c)
- static void **sqrt** (C &val, const C &c)
- static C **cbrt** (const C &c)
- static C **root** (const C &c, unsigned long k)
- static C **pow** (const C &c, const C &e)
- static C **power** (const C &c, long e)
- static C **power** (const C &c, const Z &e)
- static C **log** (const C &c)

- static void **log** (C &val, const C &c)
- static C **log2** (const C &c)
- static C **log10** (const C &c)
- static C **exp** (const C &c)
- static void **exp** (C &val, const C &c)
- static C **exp2** (const C &c)
- static C **exp10** (const C &c)
- static C **cos** (const C &c)
- static void **cos** (C &val, const C &c)
- static C **sin** (const C &c)
- static void **sin** (C &val, const C &c)
- static C **tan** (const C &c)
- static C **sec** (const C &c)
- static C **csc** (const C &c)
- static C **cot** (const C &c)
- static C **acos** (const C &c)
- static void **acos** (C &val, const C &c)
- static C **asin** (const C &c)
- static void **asin** (C &val, const C &c)
- static C **atan** (const C &c)
- static C **gamma** (const C &c)
- static C **tgamma** (const C &c)
- static C **atan2** (const C &y, const C &x)
- static C **cosh** (const C &c)

- static void **cosh** (C &val, const C &c)
- static C **sinh** (const C &c)
- static void **sinh** (C &val, const C &c)
- static C **tanh** (const C &c)
- static C **sech** (const C &c)
- static C **csch** (const C &c)
- static C **coth** (const C &c)
- static C **acosh** (const C &c)
- static C **asinh** (const C &c)
- static C **atanh** (const C &c)
- static void **conv** (C &c, const Z &z)
- static void **conv** (C &c, long n)
- static void **conv** (C &c, unsigned long u)
- static void **conv** (C &c, double d)
- static void **conv** (C &c, const R &r)

1.7.59 mere::mere_base Class Reference

Inheritance diagram for mere::mere_base:



Classes

- struct threadLocalData

Static Public Member Functions

- static size_t **defaultBits** (void)
- static size_t **defaultBits** (size_t newBits)
- static size_t **defaultDigits** (void)
- static size_t **defaultDigits** (size_t newBits)
- static bool **constantsCurrent** (size_t constantsBits)
- static void **makeConstantsCurrent** (size_t constantsBits)
- static void **free_constants** (void)
- static void **seedRandom** (void) throw ()
- static void **seedRandom** (const std::string &seed) throw ()
- static void **read** (std::istream &is, mpz_t &z_)
- static void **write** (std::ostream &os, const mpz_t &z_)
- static void **read** (std::istream &is, mpfr_t &r_, bool allowPrecision-Change=false)
- static void **write** (std::ostream &os, const mpfr_t &r_)
- static bool **anyErrorFlagSet** (void)
- static std::ostream & **abortMsgStream** (bool bAppend=false)

Protected Member Functions

- void ∗ **constantsCacheTLS_test** (void)

Static Protected Member Functions

- static threadLocalData & **threadData** (void)
- static bool **constantsCurrent** (const mere::mere_base::threadLocalData &TLS, size_t constantsBits)
- static void **makeConstantsCurrent** (mere::mere_base::threadLocalData ∗&pTLS, size_t constantBits)
- static void **free_constants_** (void ∗pData)
- static void **registerConstantsCleanup** (void)
- static void **mere_mpfr_clear** (mpfr_t f)
- static void **checkMereStatusZ** (void) throw (mere_exception)
- static void **checkMereStatusR** (void) throw (mere_exception)
- static void **checkMereStatusC** (void) throw (mere_exception)
- static unsigned int **mere_mpfr_get_flags** (void)
- static bool **underflowFlagSet** (void)
- static bool **overflowFlagSet** (void)
- static bool **nanFlagSet** (void)
- static bool **inexactFlagSet** (void)
- static bool **erangeFlagSet** (void)
- static void **seedRandom_** (size_t constantsBits) throw ()
- static void **seedRandom_** (size_t constantsBits, const std::string &seed) throw ()
- static std::string **getIntegerString** (std::istream &is)
- static std::string **getRealString** (std::istream &is)

Static Protected Attributes

- static mp_rnd_t **defaultRoundingMode_**
- static const double **log2_10_** = M_LN10/M_LN2
- static pthread_mutex_t **mere_mutex**

1.7.60 mere::mere_base::threadLocalData Struct Reference

Public Member Functions

- **threadLocalData** (size_t prec)

Public Attributes

- size_t **constantsBits_**
- gmp_randstate_t **randomState_**
- bool **normalCacheValid_**
- R ∗ **pr_normalCache_**
- R ∗ **prZero_**
- R ∗ **prEpsilon_**
- R ∗ **prMin_**
- R ∗ **prMax_**
- R ∗ **prOne_**
- R ∗ **prTwo_**
- R ∗ **prThree_**
- R ∗ **prPi_**
- R ∗ **prTwoPi_**

- R ∗ **prHalfPi_**
- R ∗ **prLog2_**
- R ∗ **prLog10_**
- C ∗ **pcZero_**
- C ∗ **pcOne_**
- C ∗ **pcOne_i_**

1.7.61 mere::mere_exception Class Reference

Public Types

- enum **exception_kind** {

  UNSPECIFIED__ = 0, **UNDERFLOW__**, **OVERFLOW__**, **NAN__**,

  INEXACT__, **ERANGE__**, **INVALID_STRING__**, **OWNERSHIP_-VIOLATION__** }

Public Member Functions

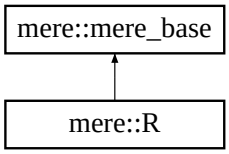
- const exception_kind & **what** (void) const
- const char ∗ **message** (void) const
- **mere_exception** (exception_kind eWhat)

Static Public Member Functions

- static const char ∗ **message** (exception_kind eWhat)

1.7.62 mere::R Class Reference

Inheritance diagram for mere::R:



Public Member Functions

- size_t **thisBits** (void) const
- size_t **thisDigits** (void) const
- int **compare** (const R &other) const
- bool **operator==** (const R &other) const
- bool **operator!=** (const R &other) const
- bool **operator**<= (const R &other) const
- bool **operator**>= (const R &other) const
- bool **operator**< (const R &other) const
- bool **operator**> (const R &other) const
- R **operator+** (void) const
- R **operator+** (const R &other) const
- R & **operator+=** (const R &other)
- R & **operator++** (void)
- R **operator++** (int)
- R **operator-** (void) const
- R **operator-** (const R &other) const

- R & **operator-=** (const R &other)
- R & **operator--** (void)
- R **operator--** (int)
- R **operator*** (const R &other) const
- R & **operator*=** (const R &other)
- R **operator/** (const R &other) const
- R & **operator/=** (const R &other)
- R & **operator+=** (const Z &z)
- R & **operator-=** (const Z &z)
- R & **operator*=** (const Z &z)
- R & **operator/=** (const Z &z)
- R & **operator=** (const Z &z)
- bool **operator==** (const Z &other) const
- bool **operator!=** (const Z &other) const
- bool **operator<=** (const Z &other) const
- bool **operator>=** (const Z &other) const
- bool **operator<** (const Z &other) const
- bool **operator>** (const Z &other) const
- bool **operator==** (long other) const
- bool **operator!=** (long other) const
- bool **operator<=** (long other) const
- bool **operator>=** (long other) const
- bool **operator<** (long other) const
- bool **operator>** (long other) const

- R & **operator=** (long other)
- R **operator+** (long other) const
- R & **operator+=** (long other)
- R **operator-** (long other) const
- R & **operator-=** (long other)
- R **operator*** (long other) const
- R & **operator*=** (long other)
- R **operator/** (long other) const
- R & **operator/=** (long other)
- void **read** (std::istream &is, bool allowPrecisionChange=false)
- void **write** (std::ostream &os) const
- bool **readBinary** (abstractCommHandle *fp)
- bool **writeBinary** (abstractCommHandle *fp) const
- size_t **binarySize** (void) const
- void **debug_print** (void) const
- void **swap** (R &other)
- void **copy** (const R &other)
- R & **operator=** (const R &other)
- R & **operator=** (const char *sR) throw (mere_exception)
- **R** (R &r, bool transferData)
- **R** (long n, void *pData, size_t precision)
- **R** (const R &r)
- **R** (const char *sR) throw (mere_exception)
- **R** (long n)

- **R** (int n)
- **R** (long n, size_t precision)

Static Public Member Functions

- static size_t **alloc_size** (size_t precision)
- static const R & **zero** (void)
- static const R & **epsilon** (void)
- static const R & **min** (void)
- static const R & **max** (void)
- static const R & **one** (void)
- static const R & **two** (void)
- static const R & **three** (void)
- static const R & **pi** (void)
- static const R & **twoPi** (void)
- static const R & **halfPi** (void)
- static const R & **log2_const** (void)
- static const R & **log10_const** (void)
- static const R & **zero** (size_t prec)
- static const R & **epsilon** (size_t prec)
- static const R & **min** (size_t prec)
- static const R & **max** (size_t prec)
- static const R & **one** (size_t prec)
- static const R & **two** (size_t prec)
- static const R & **three** (size_t prec)

- static const R & **pi** (size_t prec)
- static const R & **twoPi** (size_t prec)
- static const R & **halfPi** (size_t prec)
- static const R & **log2_const** (size_t prec)
- static const R & **log10_const** (size_t prec)
- static R **random** (void)
- static R **random** (size_t prec)
- static R **normalDist** (const R &mean, const R &variance)
- static R **normalDist** (const R &mean, const R &variance, size_t prec)
- static R **mod** (const R &num, const R &den)
- static void **mod** (R &val, const R &num, const R &den)
- static R **rem** (const R &num, const R &den)
- static void **rem** (R &val, const R &num, const R &den)
- static int **sign** (const R &r)
- static R **abs** (const R &r)
- static R **neg** (const R &r)
- static void **neg** (R &val, const R &r)
- static R **inv** (const R &r)
- static R **sqr** (const R &r)
- static R **sqrt** (const R &r)
- static R **cbrt** (const R &r)
- static R **root** (const R &r, unsigned long k)
- static R **pow** (const R &r, const R &e)
- static R **power** (const R &r, long e)

- static R **power** (const R &r, const Z &e)
- static R **log** (const R &r)
- static R **log2** (const R &r)
- static R **log10** (const R &r)
- static R **exp** (const R &r)
- static R **exp2** (const R &r)
- static R **exp10** (const R &r)
- static R **cos** (const R &r)
- static R **sin** (const R &r)
- static R **tan** (const R &r)
- static R **sec** (const R &r)
- static R **csc** (const R &r)
- static R **cot** (const R &r)
- static R **acos** (const R &r)
- static R **asin** (const R &r)
- static R **atan** (const R &r)
- static R **gamma** (const R &r)
- static R **tgamma** (const R &r)
- static R **atan2** (const R &y, const R &x)
- static R **hypot** (const R &y, const R &x)
- static void **mod2Pi** (R &r)
- static R **cosh** (const R &r)
- static R **sinh** (const R &r)
- static R **tanh** (const R &r)

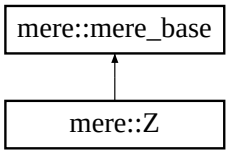
- static R **sech** (const R &r)
- static R **csch** (const R &r)
- static R **coth** (const R &r)
- static R **acosh** (const R &r)
- static R **asinh** (const R &r)
- static R **atanh** (const R &r)
- static R **euler_const** (void)
- static R **euler_const** (size_t precision)
- static R **catalan_const** (void)
- static R **catalan_const** (size_t precision)
- static R **ceil** (const R &r)
- static R **floor** (const R &r)
- static R **round** (const R &r)
- static R **trunc** (const R &r)
- static void **conv** (R &r, const Z &z)
- static void **conv** (R &r, long n)
- static void **conv** (R &r, unsigned long u)
- static void **conv** (R &r, double d)
- static void **conv** (double &d, const R &r)
- static void **conv** (long &n, const R &r)
- static void **conv** (Z &z, const R &r)

Friends

- class **mere_base::threadLocalData**
- class **C**

1.7.63 mere::Z Class Reference

Inheritance diagram for mere::Z:



Public Member Functions

- size_t **thisBits** (void) const
- size_t **thisDigits** (void) const
- int **compare** (const Z &other) const
- bool **operator==** (const Z &other) const
- bool **operator!=** (const Z &other) const
- bool **operator**<= (const Z &other) const
- bool **operator**>= (const Z &other) const
- bool **operator**< (const Z &other) const
- bool **operator**> (const Z &other) const
- Z **operator+** (void) const
- Z **operator+** (const Z &other) const

- Z & **operator+=** (const Z &other)
- Z & **operator++** (void)
- Z **operator++** (int)
- Z **operator-** (void) const
- Z **operator-** (const Z &other) const
- Z & **operator-=** (const Z &other)
- Z & **operator--** (void)
- Z **operator--** (int)
- Z **operator*** (const Z &other) const
- Z & **operator*=** (const Z &other)
- Z **operator/** (const Z &other) const
- Z & **operator/=** (const Z &other)
- Z **operator%** (const Z &other) const
- Z **operator%=** (const Z &other)
- bool **operator==** (long other) const
- bool **operator!=** (long other) const
- bool **operator<=** (long other) const
- bool **operator>=** (long other) const
- bool **operator<** (long other) const
- bool **operator>** (long other) const
- Z **operator+** (long other) const
- Z & **operator+=** (long other)
- Z **operator-** (long other) const
- Z & **operator-=** (long other)

- Z **operator**∗ (long other) const
- Z & **operator**∗**=** (long other)
- Z **operator/** (long other) const
- Z & **operator/=** (long other)
- Z **operator%** (long other) const
- Z **operator%=** (long other)
- Z & **negate** (void)
- void **read** (std::istream &is)
- void **write** (std::ostream &os) const
- bool **readBinary** (abstractCommHandle ∗fp)
- bool **writeBinary** (abstractCommHandle ∗fp) const
- size_t **binarySize** (void) const
- void **debug_print** (void) const
- void **swap** (Z &other)
- void **copy** (const Z &other)
- Z & **operator=** (const Z &other)
- Z & **operator=** (long other)
- Z & **operator=** (const char ∗sZ) throw (mere_exception)
- **Z** (Z &z, bool transferData)
- **Z** (const Z &z)
- **Z** (const char ∗sZ) throw (mere_exception)
- **Z** (long n)
- **Z** (int n)
- **Z** (long n, size_t precision)

Static Public Member Functions

- static const Z & **zero** (void)
- static const Z & **one** (void)
- static const Z & **two** (void)
- static const Z & **three** (void)
- static const Z & **zero** (size_t prec)
- static const Z & **one** (size_t prec)
- static const Z & **two** (size_t prec)
- static const Z & **three** (size_t prec)
- static Z **random** (void)
- static Z **random** (size_t prec)
- static Z **mod** (const Z &num, const Z &den)
- static Z **rem** (const Z &num, const Z &den)
- static bool **isOdd** (const Z &z)
- static bool **isEven** (const Z &z)
- static int **sign** (const Z &z)
- static Z **abs** (const Z &z)
- static Z **neg** (const Z &z)
- static Z **sqr** (const Z &z)
- static Z **power** (const Z &z, unsigned long e)
- static Z **power2** (const Z &z, long e)
- static void **conv** (Z &z, long n)
- static void **conv** (Z &z, unsigned long u)
- static void **conv** (long &n, const Z &z)
- static void **conv** (unsigned long &u, const Z &z)

Friends

- class **R**
- class **C**

1.7.64 number::numberTraits< T > Class Template Reference

traits class enforcing number system heirarchy.

1.7.64.1 Detailed Description

template<class T> class number::numberTraits< T >

traits class enforcing number system heirarchy. basic number traits are: -- typedef T complexType; // complex type if T is real. -- typedef T magnitudeType; // magnitude type if T is complex. -- typedef T integralType; // integer type of corresponding range. -- typedef T POD_VALUE_RETURN; // const reference for non-POD types

1.7.65 number::numberTraits< mere::C > Class Template Reference

Public Types

- typedef mere::C **complexType**
- typedef mere::R **magnitudeType**
- typedef mere::Z **integralType**
- typedef const mere::C & **POD_VALUE_RETURN**

template<> class number::numberTraits< mere::C >

1.7.66 number::numberTraits< mere::R > Class Template Reference

Public Types

- typedef mere::C **complexType**
- typedef mere::R **magnitudeType**
- typedef mere::Z **integralType**
- typedef const mere::R & **POD_VALUE_RETURN**

template<> class number::numberTraits< mere::R >

1.7.67 number::numberTraits< mere::Z > Class Template Reference

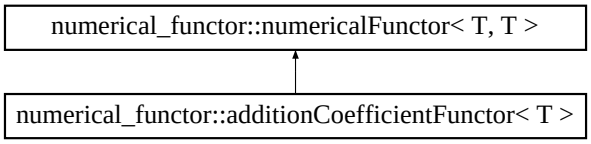
Public Types

- typedef mere::Z **magnitudeType**
- typedef mere::Z **integralType**
- typedef const mere::Z & **POD_VALUE_RETURN**

template<> class number::numberTraits< mere::Z >

1.7.68 numerical_functor::additionCoefficientFunctor< T > Class Template Reference

Inheritance diagram for numerical_functor::additionCoefficientFunctor< T >:



Classes

- class parameters
- struct **workspace**

Public Types

- typedef numberTraits< T >::magnitudeType **R**
- typedef size_t(∗ **indexFoop**)(long, long, size_t)
- typedef void(∗ **inverseIndexFoop**)(size_t, size_t, long &, long &)

Public Member Functions

- **additionCoefficientFunctor** (const additionCoefficientFunctor &other)
- additionCoefficientFunctor & **operator=** (const additionCoefficientFunctor &other)
- additionCoefficientFunctor & **copy** (const additionCoefficientFunctor &other)
- numericalFunctor< T > ∗ **clone** (void) const
- bool **setParameters** (const typename numericalFunctor< T, T >::parameters &param)
- const parameters & **getParameters** (void) const

- template<class MAT1 , class MAT2 >

  bool **apply** (const T &x, const MAT1 &aH_, const MAT2 &aK_) const throw ()

- template<class MAT >

  bool **applyScalar** (const T &x, const MAT &dest_) const throw ()

- template<class MAT >

  bool **applyScalar_l1l2** (const T &x, const MAT &dest_) const throw ()

- template<class MAT >

  bool **apply** (const T &x, const R &theta, const R &phi, const MAT &dest_) const throw ()

Protected Member Functions

- template<class MAT >

  bool **applyScalar_** (const T &x, const MAT &dest_) const throw ()

- template<class MAT >

  bool **applyScalar_l1l2_** (const T &x, const MAT &dest_) const throw ()

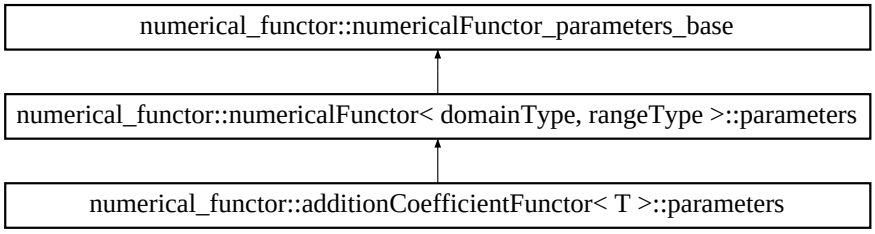
- template<class MAT >

  bool **normalize** (typename legendreFunctor< R >::parameters::normalization_kind eNorm, bool vectorNorm, bool axialOnly, const MAT &dest_) const throw ()

template<class T> class numerical_functor::additionCoefficientFunctor< T >

1.7.69 numerical_functor::additionCoefficientFunctor< T >::parameters Class Reference

Inheritance diagram for numerical_functor::additionCoefficientFunctor< T >::parameters:



Public Member Functions

- long **scalar_l_max** (void) const
- parameters & **operator=** (const parameters &other)
- parameters & **copy** (const parameters &other)
- numericalFunctor< T >::parameters * **clone** (void) const
- bool **valid** (void) const
- **parameters** (const parameters &param)

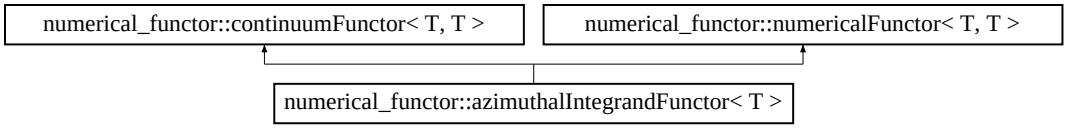
Public Attributes

- long **l1_max**
- long **l2_max**
- sbesselFunctor< T >::parameters::bessel_kind **besselKind**
- legendreFunctor< R >::parameters::normalization_kind **eNorm**
- indexFoop **index**
- inverseIndexFoop **inverseIndex**

template<class T> class numerical_functor::additionCoefficientFunctor< T >::parameters

1.7.70 numerical_functor::azimuthalIntegrandFunctor< T > Class Template Reference

Inheritance diagram for numerical_functor::azimuthalIntegrandFunctor< T >:



Classes

- class parameters
- class quadratureCacheNode

Public Types

- typedef numberTraits< T >::magnitudeType **R**

Public Member Functions

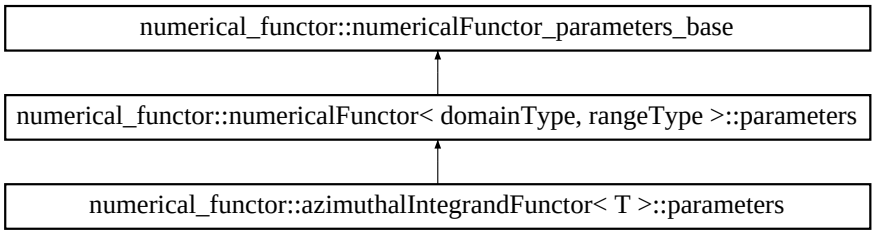
- **azimuthalIntegrandFunctor** (const azimuthalIntegrandFunctor &other)
- azimuthalIntegrandFunctor & **operator=** (const azimuthalIntegrandFunctor &other)
- azimuthalIntegrandFunctor & **copy** (const azimuthalIntegrandFunctor &other)

- numericalFunctor< T > * **clone** (void) const
- const parameters & **getParameters** (void) const
- virtual bool **apply** (const T &arg, T &val, T &fError) const throw ()
- virtual bool **apply** (const std::vector< T > &vArg, std::vector< T > &vVal, T &fError) const throw ()
- virtual bool **apply** (const std::vector< T > &vArg, typename continuumFunctor< T, T >::quadratureCacheNode *pCacheNode, std::vector< T > &vVal, T &fError) const throw ()
- virtual continuumFunctor< T, T >::quadratureCacheNode * **allocCacheNode** (const quadratureFunctor< T, T > *pOwner) const
- virtual void **setStatusString** (const char *msg) const
- virtual void **setStatusString** (const std::string &msg) const
- virtual const std::string & **getStatusString** (void) const

template<class T> class numerical_functor::azimuthalIntegrandFunctor< T >

1.7.71 numerical_functor::azimuthalIntegrandFunctor< T >::parameters Class Reference

Inheritance diagram for numerical_functor::azimuthalIntegrandFunctor< T >::parameters:



Public Member Functions

- **parameters** (const parameters &param)
- parameters & **operator=** (const parameters &other)
- parameters & **copy** (const parameters &other)
- numericalFunctor< T >::parameters * **clone** (void) const
- bool **valid** (void) const
- void **setRegion** (const regionSelector< T > &region)
- const regionSelector< T > & **region** (void) const

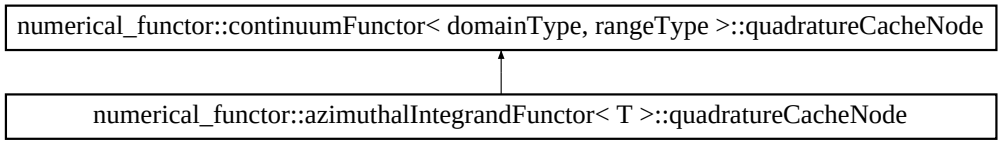
Public Attributes

- REGULARITY_KIND **eReg1**
- long **l1**
- long **l2**
- long **l3**
- long **m3**
- long **lambda**
- long **scalar_l_max**
- long **precision**

template<class T> class numerical_functor::azimuthalIntegrandFunctor< T >::parameters

1.7.72 numerical_functor::azimuthalIntegrandFunctor< T >::quadratureCacheNode Class Reference

Inheritance diagram for numerical_functor::azimuthalIntegrandFunctor< T >::quadratureCacheNode:



Public Types

- typedef numberTraits< T >::magnitudeType **R**

Public Member Functions

- **quadratureCacheNode** (const quadratureFunctor< T, T > *pOwner)

Public Attributes

- T **fError**
- std::vector< T > **vR**
- std::vector< T > **vR_s**
- std::vector< T > **vvN** [3]
- gmm::dense_matrix< T > **aBessel_interior**

- gmm::dense_matrix< T > **aBessel_exterior**
- gmm::dense_matrix< R > **aLegendre**

template<class T> class numerical_functor::azimuthalIntegrandFunctor< T >::quadratureCacheNode

1.7.73 numerical_functor::continuumFunctor< domainType, rangeType > Class Template Reference

Classes

- class quadratureCacheNode

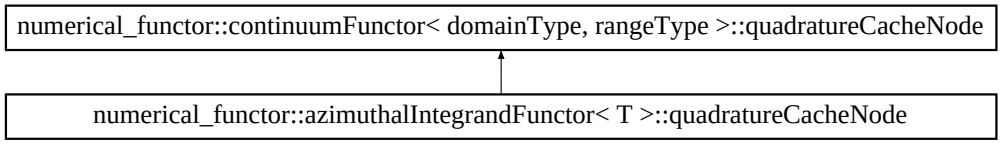
Public Member Functions

- virtual bool **apply** (const domainType &arg, rangeType &val, rangeType &fError) const =0 throw ()
- virtual bool **apply** (const std::vector< domainType > &vArg, std::vector< rangeType > &vVal, rangeType &fError) const throw ()
- virtual bool **apply** (const std::vector< domainType > &vArg, quadratureCacheNode *pCacheNode, std::vector< rangeType > &vVal, rangeType &fError) const throw ()
- virtual bool **inDomain** (const domainType &arg) const
- virtual quadratureCacheNode * **allocCacheNode** (const quadratureFunctor< domainType, rangeType > *pOwner) const
- virtual void **setStatusString** (const char *msg) const =0
- virtual void **setStatusString** (const std::string &msg) const =0
- virtual const std::string & **getStatusString** (void) const =0

template<class domainType, class rangeType> class numerical_-functor::continuumFunctor< domainType, rangeType >

1.7.74 numerical_functor::continuumFunctor< domainType, rangeType >::quadratureCacheNode Class Reference

Inheritance diagram for numerical_functor::continuumFunctor< domainType, rangeType >::quadratureCacheNode:



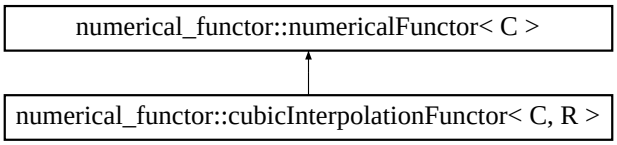
Public Member Functions

- bool **initialized** (void)
- void **setInitialized** (void)
- void **lock** (void)
- void **unlock** (void)
- **quadratureCacheNode** (const quadratureFunctor< domainType, rangeType > *pOwner)

template<class domainType, class rangeType> class numerical_-functor::continuumFunctor< domainType, rangeType >::quadratureCacheNode

1.7.75 numerical_functor::cubicInterpolationFunctor< C, R > Class Template Reference

Inheritance diagram for numerical_functor::cubicInterpolationFunctor< C, R >:



Classes

- class parameters

Public Member Functions

- numericalFunctor< C > * **clone** (void) const
- const parameters & **getParameters** (void) const
- virtual bool **updateParameters** (void)
- bool **apply** (const R &x, C &y, C &fError) const
- bool **apply** (const R &x, C &y) const
- template<class V >
  bool **apply** (const V &vX, std::vector< C > &vY, C &fError) const
- template<class V >
  bool **apply** (const V &vX, std::vector< C > &vY) const
- const R & **minArg** (void) const throw ()
- const R & **maxArg** (void) const throw ()

Protected Member Functions

- template<class U , class V >

  bool **apply_** (const U &vX, V &vY, C &fError) const
- bool **inDomain** (const R &arg) const throw ()

Static Protected Member Functions

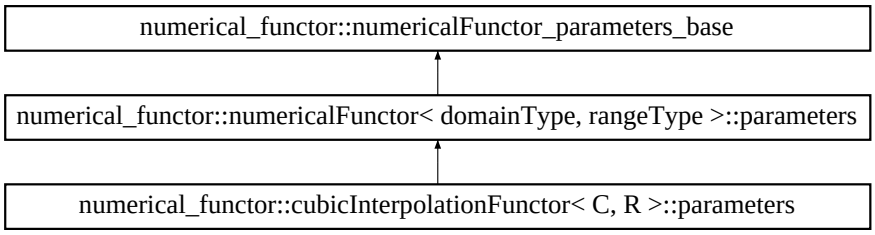
- template<class V >

  static void **powerVector** (const C &arg, V &vPow)

template<class C, class R> class numerical_-functor::cubicInterpolationFunctor< C, R >

1.7.76 numerical_functor::cubicInterpolationFunctor< C, R >::parameters Class Reference

Inheritance diagram for numerical_functor::cubicInterpolationFunctor< C, R >::parameters:



Public Types

- enum **interp_kind** { **PCHIP**, **SPLINE** }

Public Member Functions

- **parameters** (const parameters &param)
- parameters & **copy** (const parameters &other)
- numericalFunctor< C >::parameters * **clone** (void) const
- bool **valid** (void) const

Public Attributes

- long **precision**
- interp_kind **eKind**
- std::vector< C > **vDataOrdinate**
- std::vector< R > **vDataAbscissa**

template<class C, class R> class numerical_functor::cubicInterpolationFunctor< C, R >::parameters

1.7.77 numerical_functor::discreteFunctor< domainType, rangeType > Class Template Reference

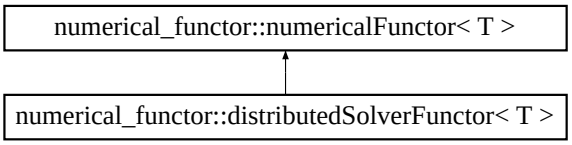
Public Member Functions

- virtual bool **apply** (const std::vector< domainType > &vArgs, rangeType &val, rangeType &fError) const =0 throw ()
- virtual bool **inDomain** (const std::vector< domainType > &vArgs) const =0

template<class domainType, class rangeType> class numerical_-functor::discreteFunctor< domainType, rangeType >

1.7.78 numerical_functor::distributedSolverFunctor< T > Class Template Reference

Inheritance diagram for numerical_functor::distributedSolverFunctor< T >:



Classes

- class parameters

Public Types

- typedef numberTraits< T >::magnitudeType **R**

Public Member Functions

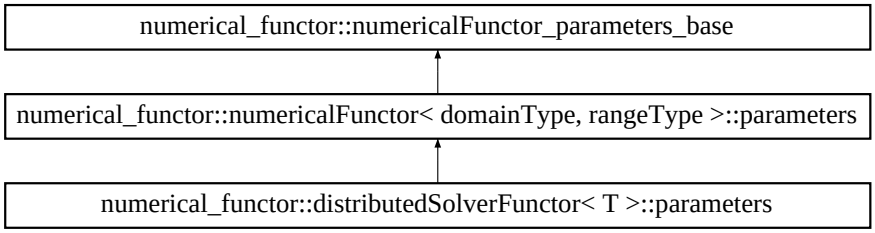
- virtual numericalFunctor< T > * **clone** (void) const
- const parameters & **getParameters** (void) const
- virtual bool **updateParameters** (void)
- void **global_block_indices** (size_t &global_row_blocks, std::vector< size_-t > &v_rb, size_t &global_col_blocks, std::vector< size_t > &v_cb) const throw ()

- bool **init_LHS_block** (const size_t rb, const size_t cb, const gmm::dense_-matrix< T > &M) throw ()
- bool **init_RHS_block** (const size_t rb, const gmm::dense_matrix< T > &M) throw ()
- bool **clear_solution_block** (const size_t rb) throw ()
- bool **factor** (void) throw ()
- bool **solve** (void) throw ()
- bool **solve_masked** (const size_t global_row_blocks, const std::vector< size_t > &v_rb, const size_t global_col_blocks, const std::vector< size_t > &v_cb, const std::vector< size_t > &vn_RHS_cols) throw ()
- size_t **max_local_row_blocks** (void) const throw ()
- bool **get_LHS_block** (const size_t rb, const size_t cb, gmm::dense_matrix< T > &M) const throw ()
- bool **get_RHS_block** (const size_t rb, gmm::dense_matrix< T > &M) const throw ()
- bool **get_solution_block** (const size_t rb, gmm::dense_matrix< T > &M) const throw ()
- bool **get_solution_columns** (const size_t rb, const std::vector< size_t > &vn_RHS_cols, gmm::dense_matrix< T > &M) const throw ()

template<class T> class numerical_functor::distributedSolverFunctor< T >

1.7.79 numerical_functor::distributedSolverFunctor< T >::parameters Class Reference

Inheritance diagram for numerical_functor::distributedSolverFunctor< T >::parameters:



Public Types

- enum **SOLVER_KIND** { **DIRECT_LU**, **BLOCK_GMRES** }

Public Member Functions

- const MPI::Comm & **comm** (void) const
- void **setComm** (const MPI::Comm &newComm)
- parameters & **copy** (const parameters &other)
- numericalFunctor< T >::parameters * **clone** (void) const
- bool **valid** (void) const
- **parameters** (const parameters &param)

Static Public Member Functions

- static size_t **max_krylov_dim** (size_t dim, size_t RHS_blocks)

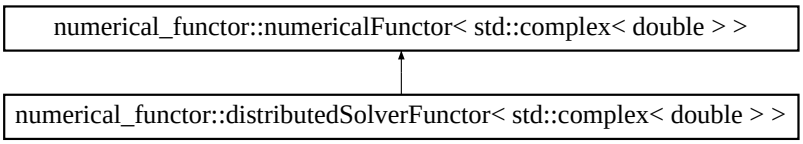
Public Attributes

- SOLVER_KIND **eSolver**
- size_t **row_blocks**
- size_t **col_blocks**
- size_t **block_nrows**
- size_t **block_ncols**
- size_t **N_RHS**
- R **convergence_tolerance**
- size_t **max_iterations**
- size_t **max_restarts**
- size_t **RHS_blocks**
- size_t **krylov_dim**
- int **verbosity**

template<class T> class numerical_functor::distributedSolverFunctor< T >::parameters

1.7.80 numerical_functor::distributedSolverFunctor< std::complex< double > > Class Template Reference

Inheritance diagram for numerical_functor::distributedSolverFunctor< std::complex< double > >:



Classes

- class parameters

Public Types

- typedef numberTraits< std::complex< double > >::magnitudeType **R**

Public Member Functions

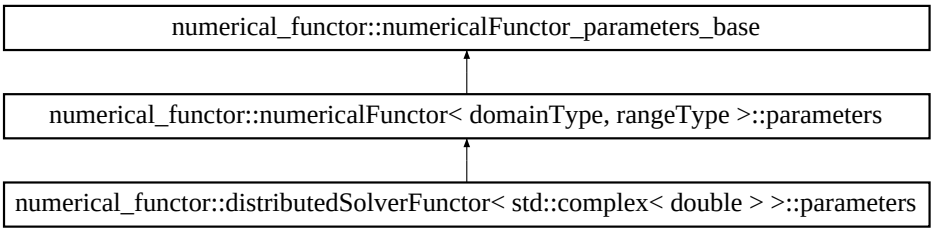
- virtual numericalFunctor< std::complex< double > > * **clone** (void) const
- const parameters & **getParameters** (void) const
- virtual bool **updateParameters** (void)
- void **global_block_indices** (size_t &global_row_blocks, std::vector< size_t > &v_rb, size_t &global_col_blocks, std::vector< size_t > &v_cb) const throw ()
- bool **init_LHS_block** (const size_t rb, const size_t cb, const gmm::dense_matrix< std::complex< double > > &M) throw ()
- bool **init_RHS_block** (const size_t rb, const gmm::dense_matrix< std::complex< double > > &M) throw ()
- bool **clear_solution_block** (const size_t rb) throw ()
- bool **factor** (void) throw ()
- bool **solve** (void) throw ()
- bool **solve_masked** (const size_t global_row_blocks, const std::vector< size_t > &v_rb, const size_t global_col_blocks, const std::vector< size_t > &v_cb, const std::vector< size_t > &vn_RHS_cols) throw ()
- size_t **max_local_row_blocks** (void) const throw ()

- bool **get_LHS_block** (const size_t rb, const size_t cb, gmm::dense_matrix< std::complex< double > > &M) const throw ()
- bool **get_RHS_block** (const size_t rb, gmm::dense_matrix< std::complex< double > > &M) const throw ()
- bool **get_solution_block** (const size_t rb, gmm::dense_matrix< std::complex< double > > &M) const throw ()
- bool **get_solution_columns** (const size_t rb, const std::vector< size_t > &vn_RHS_cols, gmm::dense_matrix< std::complex< double > > &M) const throw ()

template<> class numerical_functor::distributedSolverFunctor< std::complex< double > >

1.7.81 numerical_functor::distributedSolverFunctor< std::complex< double > >::parameters Class Reference

Inheritance diagram for numerical_functor::distributedSolverFunctor< std::complex< double > >::parameters:



Public Types

- enum **SOLVER_KIND** { **DIRECT_LU**, **BLOCK_GMRES** }

Public Member Functions

- const MPI::Comm & **comm** (void) const
- void **setComm** (const MPI::Comm &newComm)
- parameters & **copy** (const parameters &other)
- numericalFunctor< std::complex< double > >::parameters * **clone** (void) const
- bool **valid** (void) const
- **parameters** (const parameters &param)

Static Public Member Functions

- static size_t **max_krylov_dim** (size_t dim, size_t RHS_blocks)

Public Attributes

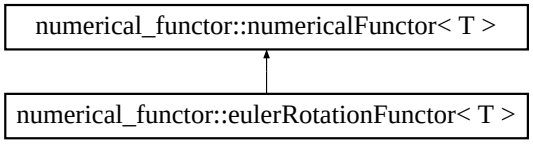
- SOLVER_KIND **eSolver**
- size_t **row_blocks**
- size_t **col_blocks**
- size_t **block_nrows**
- size_t **block_ncols**
- size_t **N_RHS**
- R **convergence_tolerance**
- size_t **max_iterations**

- size_t **max_restarts**
- size_t **RHS_blocks**
- size_t **krylov_dim**
- int **verbosity**

template<> class numerical_functor::distributedSolverFunctor< std::complex< double > >::parameters

1.7.82 numerical_functor::eulerRotationFunctor< T > Class Template Reference

Inheritance diagram for numerical_functor::eulerRotationFunctor< T >:



Classes

- class parameters

Public Member Functions

- numericalFunctor< T > * **clone** (void) const
- const parameters & **getParameters** (void) const
- virtual bool **updateParameters** (void)

- bool **apply** (const T &alpha, const T &beta, const T &gamma, T &x, T &y, T &z) const
- bool **apply** (const T &alpha, const T &beta, const T &gamma, gmm::dense_-matrix< T > &aRot, T &fError) const
- bool **apply** (const T &alpha, const T &beta, const T &gamma, gmm::dense_-matrix< T > &aRot) const
- bool **operator()** (const T &alpha, const T &beta, const T &gamma, gmm::dense_matrix< T > &aRot, T &fError) const
- bool **operator()** (const T &alpha, const T &beta, const T &gamma, gmm::dense_matrix< T > &aRot) const
- bool **inverse** (const T &alpha, const T &beta, const T &gamma, T &x, T &y, T &z) const
- bool **inverse** (const T &alpha, const T &beta, const T &gamma, gmm::dense_matrix< T > &aRot, T &fError) const
- bool **inverse** (const T &alpha, const T &beta, const T &gamma, gmm::dense_matrix< T > &aRot) const

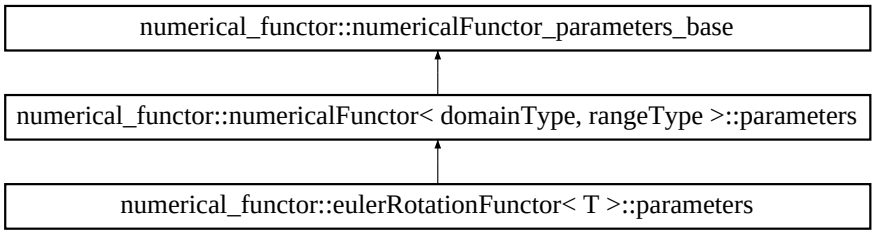
Protected Member Functions

- template<class M >
  bool **apply_** (const T &alpha, const T &beta, const T &gamma, M &aRot, T &fError) const
- template<class M >
  bool **applyInverse_** (const T &alpha, const T &beta, const T &gamma, M &aRot, T &fError) const
- bool **inDomain** (const T &alpha, const T &beta, const T &gamma) const

template<class T> class numerical_functor::eulerRotationFunctor< T >

1.7.83 numerical_functor::eulerRotationFunctor< T >::parameters Class Reference

Inheritance diagram for numerical_functor::eulerRotationFunctor< T >::parameters:



Public Member Functions

- **parameters** (const parameters &param)
- parameters & **copy** (const parameters &other)
- numericalFunctor< T >::parameters * **clone** (void) const
- bool **valid** (void) const

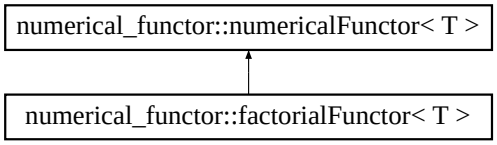
Public Attributes

- long **precision**

template<class T> class numerical_functor::eulerRotationFunctor< T >::parameters

1.7.84 numerical_functor::factorialFunctor< T > Class Template Reference

Inheritance diagram for numerical_functor::factorialFunctor< T >:



Classes

- class parameters

Public Member Functions

- numericalFunctor< T > * **clone** (void) const
- virtual bool **setParameters** (const typename numericalFunctor< T >::parameters &param)
- const parameters & **getParameters** (void) const
- bool **apply** (const long &arg, T &val, T &fError, bool bForceNoLookupTable=false) const
- bool **apply** (const long &arg, T &val) const
- const T & **apply** (const long &arg) const throw ()
- bool **apply2** (const long &argNum, const long &argDenom, T &val, T &fError, bool bForceNoLookupTable=false) const
- bool **apply2** (const long &argNum, const long &argDenom, T &val) const
- bool **inDomain** (const long &arg) const

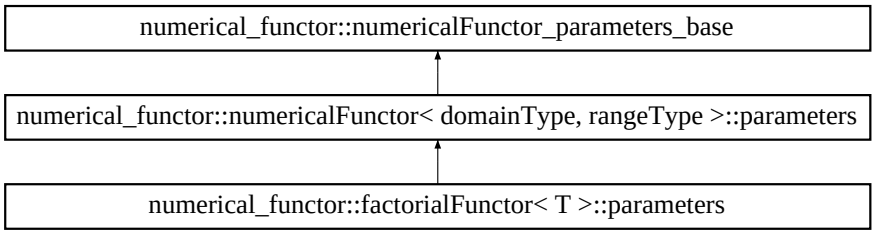
Protected Member Functions

- virtual bool **createLookupTable** (void)

template<class T> class numerical_functor::factorialFunctor< T >

1.7.85 numerical_functor::factorialFunctor< T >::parameters Class Reference

Inheritance diagram for numerical_functor::factorialFunctor< T >::parameters:



Public Member Functions

- **parameters** (const parameters &param)
- parameters & **copy** (const parameters &other)
- numericalFunctor< T >::parameters * **clone** (void) const
- bool **valid** (void) const
- virtual std::string **parentTypeName** (void) const
- virtual std::string **lookupTableAppendString** (void) const
- long **lookupTableIndex** (const long &arg) const

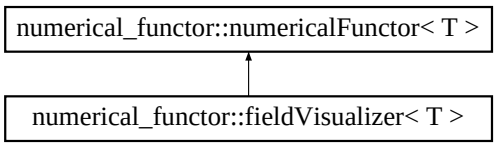
Public Attributes

- long **precision**
- long **minArg**
- long **maxArg**

template<class T> class numerical_functor::factorialFunctor< T >::parameters

1.7.86 numerical_functor::fieldVisualizer< T > Class Template Reference

Inheritance diagram for numerical_functor::fieldVisualizer< T >:



Classes

- class parameters

Public Types

- typedef numberTraits< T >::magnitudeType **R**

Public Member Functions

- numericalFunctor< T > * **clone** (void) const
- virtual bool **setParameters** (const typename numericalFunctor< T >::parameters &param)

- const parameters & **getParameters** (void) const
- virtual bool **updateParameters** (void)
- bool **apply** (const vectorFieldKet< T > &ket, std::vector< std::complex< double > > &vEr, std::vector< std::complex< double > > &vEt, std::vector< std::complex< double > > &vEp, std::vector< std::complex< double > > &vBr, std::vector< std::complex< double > > &vBt, std::vector< std::complex< double > > &vBp, const T &k=one< T >()) const throw ()
- bool **apply** (const regionSelector< T > &internalRegion, const vectorFieldKet< T > &internalFields, const vectorFieldKet< T > &externalFields, std::vector< std::complex< double > > &vEr, std::vector< std::complex< double > > &vEt, std::vector< std::complex< double > > &vEp, std::vector< std::complex< double > > &vBr, std::vector< std::complex< double > > &vBt, std::vector< std::complex< double > > &vBp) const throw ()
- bool **apply** (const std::list< regionKet< T > > &regions, std::vector< std::complex< double > > &vEr, std::vector< std::complex< double > > &vEt, std::vector< std::complex< double > > &vEp, std::vector< std::complex< double > > &vBr, std::vector< std::complex< double > > &vBt, std::vector< std::complex< double > > &vBp) const throw ()
- const std::vector< double > **vXX** (void) const
- const std::vector< double > **vYY** (void) const
- const std::vector< double > **vZZ** (void) const
- void **get_vNN** (const regionSelector< T > &internalRegion, std::vector< double > &vNN_r, std::vector< double > &vNN_t, std::vector< double > &vNN_p) const

- bool **apply** (const vectorFieldKet< T > &ket, std::vector< std::complex< double > > &vVSW_M, std::vector< std::complex< double > > &vVSW_-N, const long &m_, REGULARITY_KIND eReg) const throw ()
- bool **showRegions** (const std::list< regionKet< T > > &regions, std::vector< long > &vLabels) const throw ()

Static Public Member Functions

- static void **subtask_mesh** (size_t rank, size_t N_proc, const parameters &globalParam, parameters &subtaskParam) throw ()
- static void **subtask_mesh_** (size_t rank, size_t N_proc, const std::vector< size_t > &shape, std::vector< std::pair< size_t, size_t > > &intervals)
- static void **alloc_mesh** (const parameters &param, std::vector< long > &vLabels, std::vector< double > &vXX, std::vector< double > &vYY, std::vector< double > &vZZ, std::vector< std::complex< double > > &vEr, std::vector< std::complex< double > > &vEt, std::vector< std::complex< double > > &vEp, std::vector< std::complex< double > > &vBr, std::vector< std::complex< double > > &vBt, std::vector< std::complex< double > > &vBp) throw ()
- static void **combine_to_global_mesh** (size_t rank, size_t N_proc, const parameters &globalParam, const std::vector< long > &vLabels_sub, const std::vector< double > &vXX_sub, const std::vector< double > &vYY_sub, const std::vector< double > &vZZ_sub, const std::vector< std::complex< double > > &vEr_sub, const std::vector< std::complex< double > > &vEt_sub, const std::vector< std::complex< double > > &vEp_sub, const std::vector< std::complex< double > > &vBr_sub, const std::vector< std::complex< double > > &vBt_sub, const std::vector< std::complex<

double > > &vBp_sub, std::vector< long > &vLabels, std::vector< double > &vXX, std::vector< double > &vYY, std::vector< double > &vZZ, std::vector< std::complex< double > > &vEr, std::vector< std::complex< double > > &vEt, std::vector< std::complex< double > > &vEp, std::vector< std::complex< double > > &vBr, std::vector< std::complex< double > > &vBt, std::vector< std::complex< double > > &vBp) throw ()

Protected Member Functions

- void **createMesh** (void)
- void **recenterMesh** (const ntuple< R, 3 > &center) const
- template<class U >
  const U **minNotZero** (const U &a, const U &b) const
- template<class U >
  const U **minNotZero** (const U &a, const U &b, const U &c) const
- void **createUniformDomain** (const T &min_, const T &max_, const T &delta_, std::vector< T > &vT) const
- void **createUniformDomainMod** (const T &min_, const T &max_, const T &delta_, const T &modulus_, std::vector< T > &vT) const

Protected Attributes

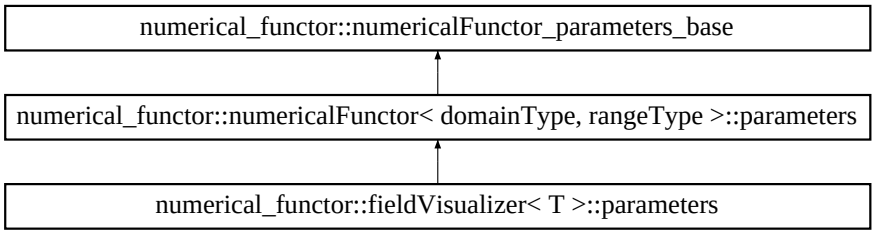
- std::vector< T > **vR**
- std::vector< T > **vT**
- std::vector< T > **vP**
- std::vector< T > **vU**
- std::vector< R > **vXX_**

- std::vector< R > **vYY_**
- std::vector< R > **vZZ_**
- std::vector< unsigned long > **vnRR**
- std::vector< unsigned long > **vnTT**
- std::vector< unsigned long > **vnPP**
- bool **rotate_**
- gmm::dense_matrix< R > **aRot_**
- gmm::dense_matrix< R > **aInvRot_**

template<class T> class numerical_functor::fieldVisualizer< T >

1.7.87 numerical_functor::fieldVisualizer< T >::parameters Class Reference

Inheritance diagram for numerical_functor::fieldVisualizer< T >::parameters:



Public Member Functions

- **parameters** (const parameters &param)
- parameters & **copy** (const parameters &other)
- numericalFunctor< T >::parameters * **clone** (void) const

- bool **valid** (void) const
- virtual std::string **parentTypeName** (void) const
- bool **writeBinary** (abstractCommHandle ∗fp) const
- bool **readBinary** (abstractCommHandle ∗fp)
- void **write** (std::ostream &os) const

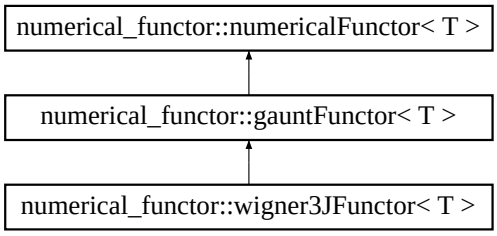
Public Attributes

- R **xMin**
- R **xMax**
- R **yMin**
- R **yMax**
- R **zMin**
- R **zMax**
- R **alpha**
- R **beta**
- R **gamma**
- unsigned long **nX**
- unsigned long **nY**
- unsigned long **nZ**
- bool **jitter**
- R **jitterFactor**
- bool **outputJitteredMesh**
- long **precision**
- bool **cartesian**

template<class T> class numerical_functor::fieldVisualizer< T >::parameters

1.7.88 numerical_functor::gauntFunctor< T > Class Template Reference

Inheritance diagram for numerical_functor::gauntFunctor< T >:



Classes

- class parameters

Public Types

- typedef numberTraits< T >::integralType **Z**

Public Member Functions

- numericalFunctor< T > * **clone** (void) const
- const parameters & **getParameters** (void) const
- virtual bool **updateParameters** (void)
- bool **apply** (const long &l1, const long &l2, const long &l3, const long &m1, const long &m2, const long &m3, T &val, T &fError, bool

bForceNoLookupTable=false) const throw ()

- bool **apply** (const long &l1, const long &l2, const long &l3, const long &m1, const long &m2, const long &m3, T &val) const throw ()
- const T & **operator()** (const long &l1, const long &l2, const long &l3, const long &m1, const long &m2, const long &m3) const throw ()
- bool **apply** (const multipoleSextuple &s6, T &val, T &fError, bool bForceNoLookupTable=false) const throw ()
- bool **apply** (const multipoleSextuple &s6, T &val) const throw ()

Static Public Member Functions

- static bool **isTrivialZero** (const multipoleSextuple &s6Arg)
- static bool **isTrivialZero** (const long &l1, const long &l2, const long &l3, const long &m1, const long &m2, const long &m3)

Protected Member Functions

- bool **triangleCoefficient** (const long &j1, const long &j2, const long &j3, T &val) const
- bool **inDomain** (const multipoleSextuple &s6Arg) const
- long **tableIndex** (const multipoleSextuple &s6Arg) const
- virtual bool **createLookupTable** (void)
- factorialFunctor< T >::parameters **standardFactorialParameters** (void) const
- factorialFunctor< Z >::parameters **standardFactorialZParameters** (void) const

Static Protected Member Functions

- static bool **triangleCondition** (const long &l1, const long &l2, const long &l3)
- static long **tableIndex** (const gauntFunctor< T >::parameters &param, const numericalFunctorLookupTable< T > &lookupTable, const multipoleSextuple &s6Arg)

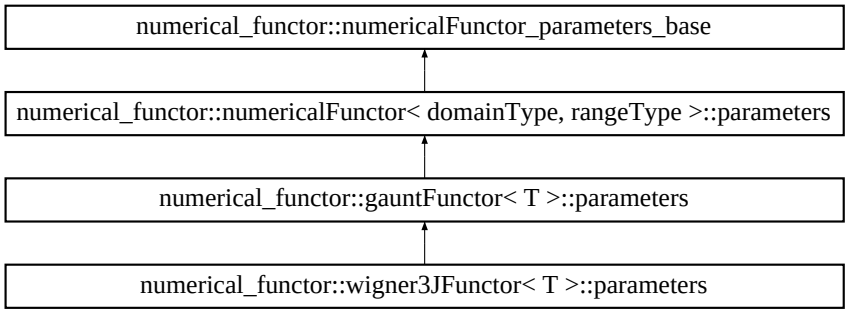
Protected Attributes

- factorialFunctor< T > **factorial**
- factorialFunctor< Z > **factorialZ**

template<class T> class numerical_functor::gauntFunctor< T >

1.7.89 numerical_functor::gauntFunctor< T >::parameters Class Reference

Inheritance diagram for numerical_functor::gauntFunctor< T >::parameters:



Public Member Functions

- **parameters** (const parameters &param)
- parameters & **copy** (const parameters &other)
- numericalFunctor< T >::parameters * **clone** (void) const
- bool **valid** (void) const
- virtual std::string **parentTypeName** (void) const
- virtual std::string **lookupTableAppendString** (void) const
- bool **setFirstTable** (void)
- bool **setNextTable** (void)
- bool **setMatchingTable** (const multipoleSextuple &s6)
- virtual bool **evenLSumRule** (void) const

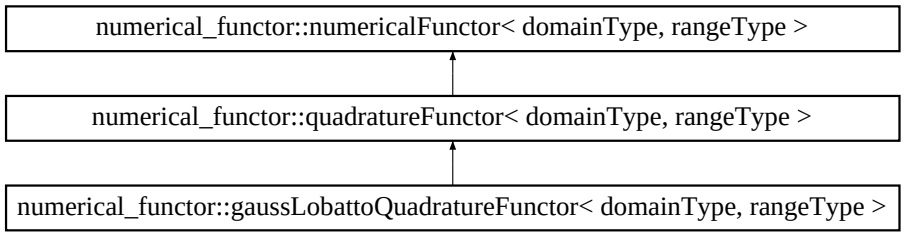
Public Attributes

- long **precision**
- unsigned long **appx_coeffPerTable**
- bool **shadowTable**
- bool **consolidateTable**
- unsigned long **consolidateFrom_coeffPerTable**
- bool **checkInDomain**
- bool **checkTrivialZero**
- bool **assumeStandardForm**
- multipoleSextuple **s6Min**
- multipoleSextuple **s6Max**

template<class T> class numerical_functor::gauntFunctor< T >::parameters

1.7.90 numerical_functor::gaussLobattoQuadratureFunctor< domainType, rangeType > Class Template Reference

Inheritance diagram for numerical_functor::gaussLobattoQuadratureFunctor< domainType, rangeType >:



Public Types

- typedef number::numberTraits< domainType >::magnitudeType **R**

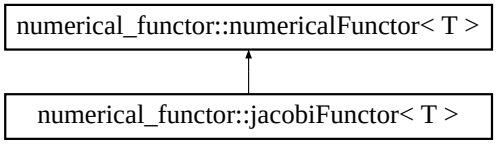
Public Member Functions

- numericalFunctor< domainType, rangeType > * **clone** (void) const
- virtual bool **apply** (const continuumFunctor< domainType, rangeType > &integrandFunctor, const domainType &intervalStart, const domainType &intervalEnd, rangeType &val, rangeType &fError) const

template<class domainType, class rangeType = domainType> class numerical_functor::gaussLobattoQuadratureFunctor< domainType, rangeType >

1.7.91 numerical_functor::jacobiFunctor< T > Class Template Reference

Inheritance diagram for numerical_functor::jacobiFunctor< T >:



Classes

- class parameters

Public Member Functions

- numericalFunctor< T > * **clone** (void) const
- const parameters & **getParameters** (void) const
- virtual bool **updateParameters** (void)
- bool **apply** (const T &x, std::vector< T > &vP, std::vector< T > &vDP, T &fError) const
- bool **apply** (const T &x, std::vector< T > &vP, std::vector< T > &vDP) const
- bool **apply** (const T &x, std::vector< T > &vP, T &fError) const
- bool **apply** (const T &x, std::vector< T > &vP) const
- bool **apply** (const std::vector< T > &vX, gmm::dense_matrix< T > &aP, gmm::dense_matrix< T > &aDP, T &fError) const

- bool **apply** (const std::vector< T > &vX, gmm::dense_matrix< T > &aP, gmm::dense_matrix< T > &aDP) const
- bool **apply** (const std::vector< T > &vX, gmm::dense_matrix< T > &aP, T &fError) const
- bool **apply** (const std::vector< T > &vX, gmm::dense_matrix< T > &aP) const

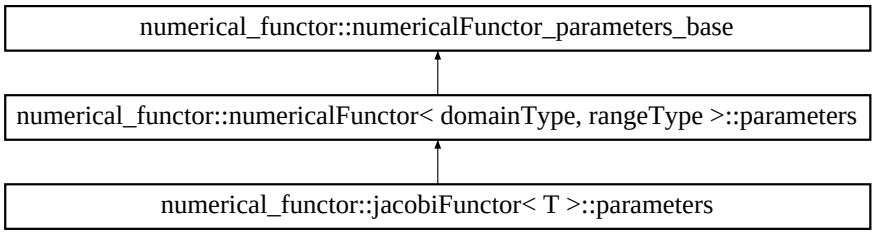
Protected Member Functions

- template<typename V >
  bool **apply_** (const T &x, V &vP, V &vDP, T &fError) const
- template<typename V >
  bool **apply_** (const T &x, V &vP, T &fError) const

template<typename T> class numerical_functor::jacobiFunctor< T >

1.7.92 numerical_functor::jacobiFunctor< T >::parameters Class Reference

Inheritance diagram for numerical_functor::jacobiFunctor< T >::parameters:



Public Member Functions

- **parameters** (const parameters &param)
- parameters & **copy** (const parameters &other)
- numericalFunctor< T >::parameters * **clone** (void) const
- bool **valid** (void) const

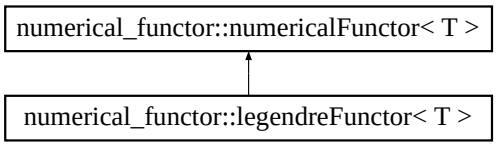
Public Attributes

- T **alpha**
- T **beta**
- long **n_max**
- long **precision**
- T **minX**
- T **maxX**

template<typename T> class numerical_functor::jacobiFunctor< T >::parameters

1.7.93 numerical_functor::legendreFunctor< T > Class Template Reference

Inheritance diagram for numerical_functor::legendreFunctor< T >:



Classes

- class parameters

Public Member Functions

- numericalFunctor< T > * **clone** (void) const
- const parameters & **getParameters** (void) const
- virtual bool **updateParameters** (void)
- bool **apply** (const T &x, std::vector< T > &vPlm, std::vector< T > &vDPlm, T &fError) const
- bool **apply** (const T &x, std::vector< T > &vPlm, std::vector< T > &vDPlm) const
- bool **apply** (const T &x, std::vector< T > &vPlm, T &fError) const
- bool **apply** (const T &x, std::vector< T > &vPlm) const
- bool **apply** (const std::vector< T > &vX, gmm::dense_matrix< T > &aPlm, gmm::dense_matrix< T > &aDPlm, T &fError) const
- bool **apply** (const std::vector< T > &vX, gmm::dense_matrix< T > &aPlm, gmm::dense_matrix< T > &aDPlm) const
- bool **apply** (const std::vector< T > &vX, gmm::dense_matrix< T > &aPlm, T &fError) const
- bool **apply** (const std::vector< T > &vX, gmm::dense_matrix< T > &aPlm) const
- template<typename V >
  void **normalize** (V &dest) const throw ()
- template<class C >
  bool **chebyshevPoints** (std::vector< C > &V, bool bCosTheta=true) const

Protected Member Functions

- template<typename V >

  void **normalize_** (V &dest) const
- bool **computeNormalization** (void)
- bool **commonNormalizationFactor** (T &val) const
- const T & **commonNormalizationFactor** (void) const
- template<typename V >

  bool **apply_** (const T &x, V &vPlm, V &vDPlm, T &fError) const
- template<typename V >

  bool **apply_** (const T &x, V &vPlm, T &fError) const

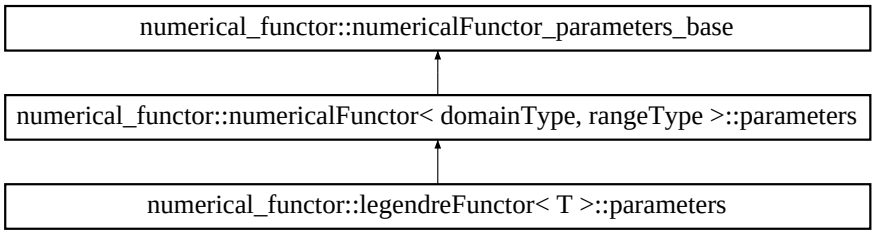
Protected Attributes

- T **commonNormalizationFactor_**
- std::vector< T > **vNorm_**

template<typename T> class numerical_functor::legendreFunctor< T >

1.7.94 numerical_functor::legendreFunctor< T >::parameters Class Reference

Inheritance diagram for numerical_functor::legendreFunctor< T >::parameters:



Public Types

- enum **normalization_kind** { **none**, **CS_phase_norm**, **full_norm** }
- enum **derivative_normalization_kind** { **sin_theta2**, **d_theta** }

Public Member Functions

- **parameters** (const parameters &param)
- parameters & **copy** (const parameters &other)
- numericalFunctor< T >::parameters * **clone** (void) const
- bool **valid** (void) const

Public Attributes

- long **m**
- long **lMax**
- long **precision**
- T **minX**
- T **maxX**
- normalization_kind **eNormalization**
- derivative_normalization_kind **eDerivativeNormalization**

template<typename T> class numerical_functor::legendreFunctor< T >::parameters

1.7.95 numerical_functor::multipolePair Class Reference

Classes

- struct **threadLocalData**

Public Member Functions

- **multipolePair** (const multipolePair &other)
- **multipolePair** (const long &l__, const long &m__, bool mIndices=false)
- bool **operator==** (const multipolePair &other) const
- long **l** (void) const
- long **m** (void) const
- long & **l** (void)
- long & **n_m** (void)
- const long & **n_m** (void) const
- void **setFrom_mValue** (const long &l__, const long &m__)
- void **setFrom_mIndex** (const long &l__, const long &n_m__)
- void **write** (std::ostream &os) const
- void **read** (std::istream &is)

Static Public Member Functions

- static unsigned long **N_m** (void)
- static unsigned long **N_mValues** (void)

- static void **setMTable** (const std::vector< long > &vM)
- static const std::vector< long > & **vM** (void)
- static bool **sparseM** (void)
- static void **setSparseM** (bool flag)
- static long **indexToValue** (const long &n_m__)
- static long **valueToIndex** (const long &m__)
- static const long & **min_mValue** (void)
- static const long & **max_mValue** (void)

1.7.96 numerical_functor::multipoleSextuple Class Reference

Classes

- class permutation

Public Member Functions

- **multipoleSextuple** (const multipoleSextuple &other)
- **multipoleSextuple** (const multipolePair &l1m1, const multipolePair &l2m2, const multipolePair &l3m3)
- **multipoleSextuple** (const long &l1, const long &l2, const long &l3, const long &m1, const long &m2, const long &m3, bool mIndices=false)
- void **setFromValues** (const long &l1, const long &l2, const long &l3, const long &m1, const long &m2, const long &m3, bool mIndices=false)
- bool **isStandardForm** (void) const
- bool **operator==** (const multipoleSextuple &other) const

- bool **sameL** (const multipoleSextuple &other) const
- bool **sameM** (const multipoleSextuple &other) const
- bool **operator!=** (const multipoleSextuple &other) const
- bool **operator<=** (const multipoleSextuple &other) const
- bool **operator>=** (const multipoleSextuple &other) const
- bool **operator<** (const multipoleSextuple &other) const
- bool **operator>** (const multipoleSextuple &other) const
- multipoleSextuple & **operator++** (void)
- multipoleSextuple **operator++** (int)
- multipoleSextuple & **operator--** (void)
- multipoleSextuple **operator--** (int)
- multipolePair & **operator[ ]** (long n)
- const multipolePair & **operator[ ]** (long n) const
- long **n_m1Max** (void) const
- long **n_m1Min** (void) const
- long **n_m2Max** (void) const
- long **n_m2Min** (void) const
- long **n_m3Max** (void) const
- long **n_m3Min** (void) const
- bool **M_inDomain** (void) const
- permutation **begin** (void) const
- permutation **end** (void) const
- permutation **beginConstantM** (void) const
- permutation **endConstantM** (void) const

- permutation **begin2** (void) const
- permutation **end2** (void) const
- permutation **begin2ConstantM** (void) const
- permutation **end2ConstantM** (void) const
- multipoleSextuple **standardForm** (void) const
- void **debug_print** (void) const
- void **write** (std::ostream &os) const
- void **read** (std::istream &is)
- bool **incrementM** (void)
- void **setM_top** (void)
- bool **incrementL** (void)
- bool **decrementM** (void)
- void **setM_bottom** (void)
- bool **decrementL** (void)

Static Public Member Functions

- static unsigned long **N_m2** (void)
- static unsigned long **N_m3** (void)
- static long **sparse_n_m1Max** (void)
- static long **sparse_n_m1Min** (void)
- static long **sparse_n_m2Max** (void)
- static long **sparse_n_m2Min** (void)
- static long **sparse_n_m3Max** (void)
- static long **sparse_n_m3Min** (void)

- static void **setIterationLimits** (const multipoleSextuple &sMin, const multipoleSextuple &sMax, bool evenLSum_=true, bool iterateWithConstantM_=false, bool iterateForKetProduct_=false)
- static void **getIterationLimits** (multipoleSextuple &sMin, multipoleSextuple &sMax)
- static void **setIterationLimitsUnrestricted** (void)
- static void **setIterationLimitsEvenLSum** (bool flag=true)
- static void **setIterationLimitsConstantM** (bool flag=true)
- static void **setIterationLimitsKetProduct** (bool flag=true)
- static bool **getIterationLimitsEvenLSum** (void)
- static bool **getIterationLimitsConstantM** (void)
- static bool **getIterationLimitsKetProduct** (void)
- static const multipoleSextuple & **iterationMin** (void)
- static const multipoleSextuple & **iterationMax** (void)
- static void **saveIterationLimits** (void)
- static void **restoreIterationLimits** (void)
- static unsigned long **numberNonzeroGauntCoefficients** (const multipoleSextuple sMax)
- static multipoleSextuple **inverseNumberNonzeroGauntCoefficients** (const unsigned long N)

Protected Attributes

- rawMultipoleSextuple **vPair**

1.7.97 numerical_functor::multipoleSextuple::permutation Class Reference

Classes

- struct **threadLocalData**

Public Types

- enum **permutation_masks** {

  degenerate2DMask = 0xff00, **degenerate3DMask** = 0x00ff, **degenerate-SextupleFlag** = 0x8, **degenerateMFlag** = 0x4,

  degenerateTripletFlag = 0x2, **degeneratePairFlag** = 0x1, **degenerateSex-tupleFlag2** = 0x8<<8, **degenerateMFlag2** = 0x4<<8,

  degeneratePairFlag2 = 0x1<<8, **npermute3TableSize** = 14, **nper-mute2TableSize** = 6, **npermute3Begin** = 1,

  npermute3End = npermute3TableSize-1, **npermute3EndConstantM** = npermute3TableSize-7, **npermute2Begin** = 1, **npermute2End** = npermute2TableSize-1,

  npermute2EndConstantM = npermute2TableSize-3 }

- enum **permutation_Kind** {

  not_a_permutation = -3, **standard_form** = -2, **finished** = -1, **none** = 0,

  cyclic1 = none, **cyclic2** = 1, **cyclic3** = 2, **anticyclic1** = 3,

  anticyclic2 = 4, **anticyclic3** = 5, **swap12** = anticyclic1, **swap23** = anticyclic2,

  swap13 = anticyclic3, **none_M** = 6, **cyclic1_M** = none_M, **cyclic2_M** = 7,

  cyclic3_M = 8, **anticyclic1_M** = 9, **anticyclic2_M** = 10, **anticyclic3_M** = 11,

swap12_M = anticyclic1_M, **swap23_M** = anticyclic2_M, **swap13_M** = anticyclic3_M, **N_permutation_Kind** = anticyclic3_M+1 }

- enum **permutation_Degeneracy** {

standard_form_D = 0, **finished_D** = 0, **none_D** = degenerateSextupleFlag, **cyclic1_D** = none_D,

cyclic2_D = degenerateTripletFlag|degenerateSextupleFlag, **cyclic3_-D** = degenerateTripletFlag|degenerateSextupleFlag, **anticyclic1_-D** = degeneratePairFlag|degenerateSextupleFlag, **anticyclic2_D** = degeneratePairFlag|degenerateSextupleFlag,

anticyclic3_D = degeneratePairFlag|degenerateSextupleFlag, **swap12_D** = anticyclic1_D, **swap23_D** = anticyclic2_D, **swap13_D** = anticyclic3_D,

none_M_D = degenerateMFlag|degenerateSextupleFlag, **cyclic1_M_D** = none_M_D, **cyclic2_M_D** = degenerateMFlag|degenerateTripletFlag|degenerateSextupleFlag, **cyclic3_-M_D** = degenerateMFlag|degenerateTripletFlag|degenerateSextupleFlag,

anticyclic1_M_D = degenerateMFlag|degeneratePairFlag|degenerateSextupleFlag, **anticyclic2_M_D** = degenerateMFlag|degeneratePairFlag|degenerateSextupleFlag, **anticyclic3_M_D** = degenerateMFlag|degeneratePairFlag|degenerateSextupleFlag, **swap12_M_D** = anticyclic1_M_D,

swap23_M_D = anticyclic2_M_D, **swap13_M_D** = anticyclic3_-M_D, **none_D2** = degenerateSextupleFlag2, **none_M_D2** = degenerateMFlag2|degenerateSextupleFlag2,

anticyclic2_D2 = degeneratePairFlag2|degenerateSextupleFlag2, **anticyclic2_M_D2** = degenerateMFlag2|degeneratePairFlag2|degenerateSextupleFlag2, **swap23_D2** = anticyclic2_D2, **swap23_M_D2** = anticyclic2_M_D2 }

- typedef long **indexTriplet** [3]

Public Member Functions

- **permutation** (const multipoleSextuple &t, bool permute2=false, bool atEnd=false)
- **permutation** (const multipoleSextuple &t, unsigned long nPermutation, bool permute2=false) throw ()
- **permutation** (const permutation &other)
- permutation & **operator++** (void)
- permutation **operator++** (int)
- permutation & **operator--** (void)
- permutation **operator--** (int)
- bool **operator==** (const permutation &other) const
- bool **operator!=** (const permutation &other) const
- bool **operator**<= (const permutation &other) const
- bool **operator**>= (const permutation &other) const
- bool **operator**< (const permutation &other) const
- bool **operator**> (const permutation &other) const
- const multipoleSextuple & **operator**∗ (void) const
- const multipoleSextuple & **operator()** (permutation_Kind eKind=none) const
- bool **apply** (multipoleSextuple &dest) const
- const multipoleSextuple & **standardForm** (void) const
- const multipoleSextuple & **standardForm** (permutation_Kind &eKind) const

Static Public Member Functions

- static void **setDegenerateConsiderMManifold** (bool flag=true)
- static bool **degenerateConsiderMManifold** (void)

Protected Member Functions

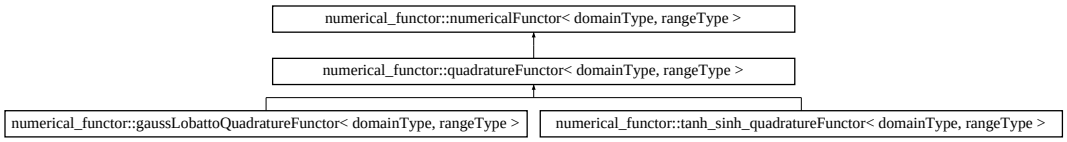
- const indexTriplet & **permuteIndices** (permutation_Kind eKind) const
- permutation_Kind **indicesToPermutation** (const indexTriplet &vI, bool mirrorM) const throw ()
- const indexTriplet & **sortedLIndices** (void) const
- bool **mirrorM** (void) const
- bool **degeneratePermute** (void) const
- void **setDegeneracyFlags** (void)

Static Protected Member Functions

- static void **permuteIndices** (permutation_Kind eKind, indexTriplet &vI)
- static long **indicesToLong** (const indexTriplet &vI, bool mirrorM)
- static bool **mirrorM** (permutation_Kind eKind)

1.7.98 numerical_functor::numericalFunctor< domainType, rangeType > Class Template Reference

Inheritance diagram for numerical_functor::numericalFunctor< domainType, rangeType >:



Classes

- class parameters

Public Member Functions

- **numericalFunctor** (unsigned long nArgumentDimensions=1)
- **numericalFunctor** (const numericalFunctor &other)
- numericalFunctor & **copy** (const numericalFunctor &other)
- virtual numericalFunctor * **clone** (void) const
- virtual bool **setParameters** (const parameters &param)
- const parameters & **getParameters** (void) const
- virtual bool **updateParameters** (void)
- bool **validParameters** (void) const
- void **setStatusString** (const char *msg) const
- void **setStatusString** (const std::string &msg) const
- const std::string & **getStatusString** (void) const
- void **unloadResidentLookupTables** (void)

Protected Member Functions

- virtual bool **createLookupTable** (void)

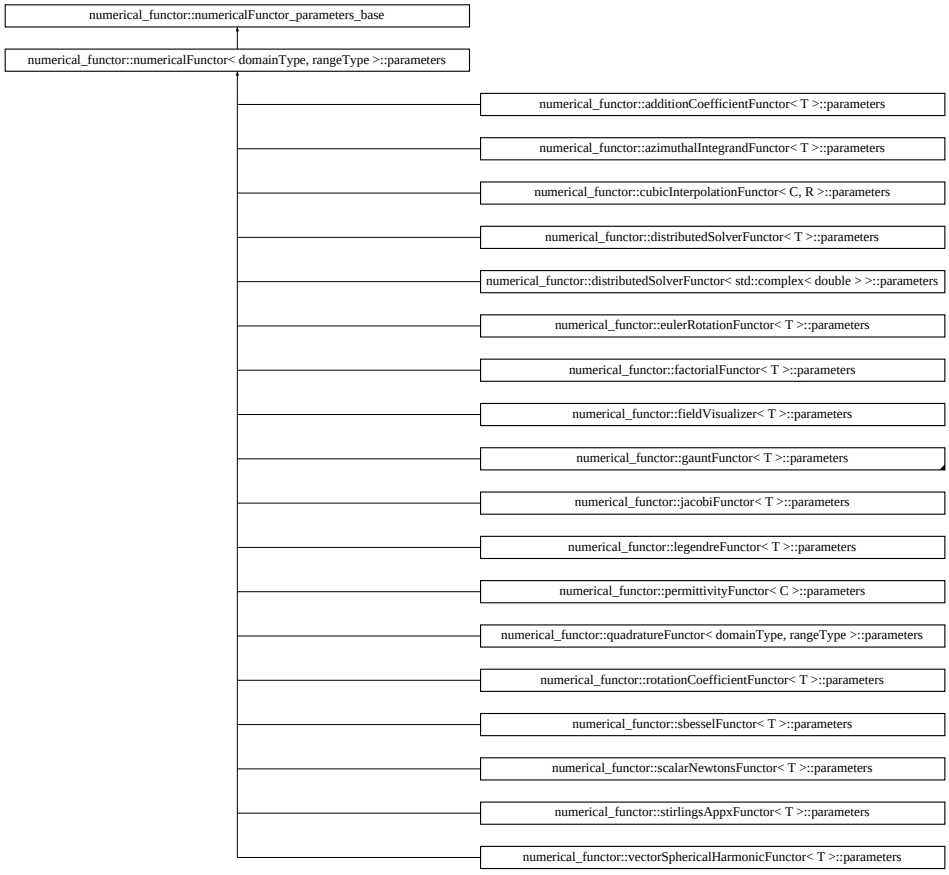
Protected Attributes

- numericalFunctorLookupTable< domainType, rangeType > **lookupTable**

template<class domainType, class rangeType = domainType> class numerical_functor::numericalFunctor< domainType, rangeType >

1.7.99 numerical_functor::numericalFunctor< domainType, rangeType >::parameters Class Reference

Inheritance diagram for numerical_functor::numericalFunctor< domainType, rangeType >::parameters:



Public Member Functions

- **parameters** (const parameters &other)
- parameters & **copy** (const parameters &other)
- virtual parameters * **clone** (void) const
- virtual bool **valid** (void) const
- virtual std::string **parentTypeName** (void) const
- virtual std::string **lookupTableAppendString** (void) const
- const std::string & **slookupTableDirectoryBase** (void) const

- const std::string & **slookupTableDirectoryOut** (void) const
- const std::string & **slookupTableListFileName** (void) const

Public Attributes

- bool **useLookupTable**
- long **numberResidentLookupTables**

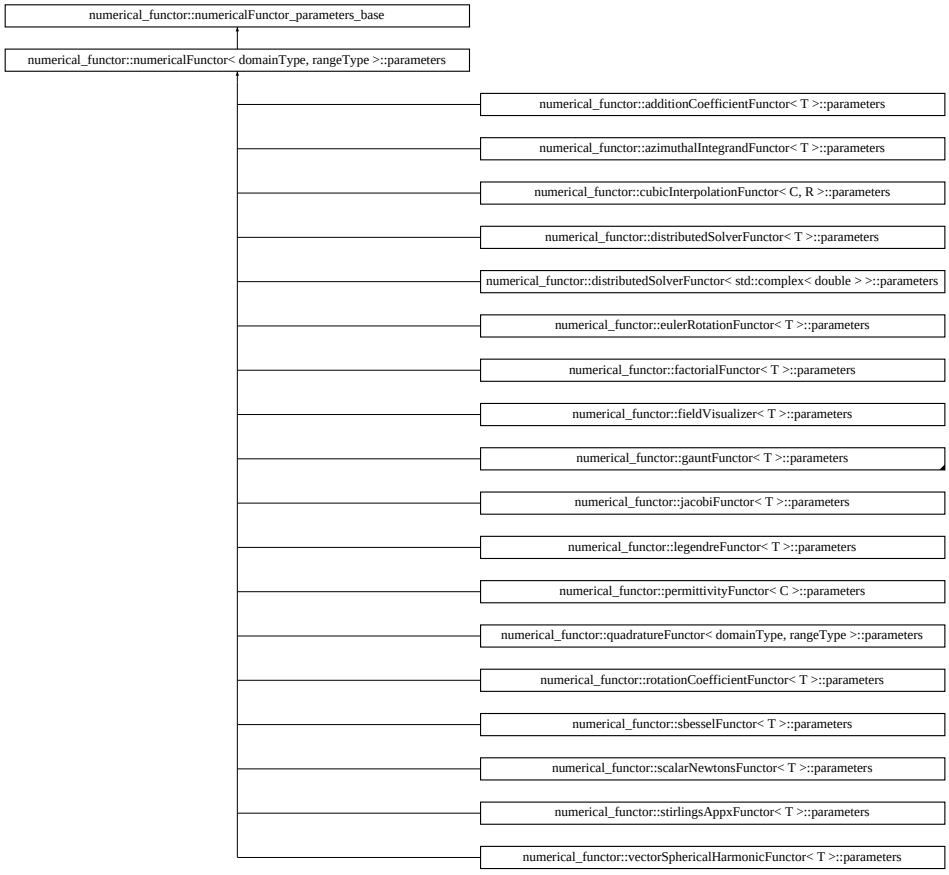
Protected Member Functions

- parameters & **operator=** (const parameters &other)

template<class domainType, class rangeType = domainType> class numerical_functor::numericalFunctor< domainType, rangeType >::parameters

1.7.100 numerical_functor::numericalFunctor_parameters_base Class Reference

Inheritance diagram for numerical_functor::numericalFunctor_parameters_base:



Static Public Member Functions

- static const std::string & **slookupTableDirectoryBase** (void)
- static const std::string & **slookupTableDirectoryOut** (void)
- static const std::string & **slookupTableListFileName** (void)

Static Protected Member Functions

- static bool **initialized** (void)
- static void **initStatics** (void)

1.7.101 numerical_functor::numericalFunctorLookupTable< domainType, rangeType > Class Template Reference

Classes

- class tableNode

Public Member Functions

- **numericalFunctorLookupTable** (const long &nDim=1, const long &nResident=2)
- **numericalFunctorLookupTable** (const numericalFunctorLookupTable &other)
- bool **lookupTableExists** (const typename numericalFunctor< domainType, rangeType >::parameters &param) const
- bool **current** (const typename numericalFunctor< domainType, rangeType >::parameters &param) const
- bool **swapResidentTables** (const typename numericalFunctor< domainType, rangeType >::parameters &param)
- void **addResidentTable** (const typename numericalFunctor< domainType, rangeType >::parameters &param)
- const std::vector< rangeType > & **table** (void) const
- const std::vector< std::vector< long > > & **indexVectors** (void) const throw ()
- const unsigned long & **dimensions** (void) const
- long **index** (const std::vector< long > &vX) const throw ()

- long **index** (const std::vector< long > &vIndex, const std::vector< long > &vBase) const throw ()
- long **index** (const long &x0, const long &x1, const long &x2, const long &b0, const long &b1, const long &b2) const throw ()
- bool **loadTable** (const typename numericalFunctor< domainType, rangeType >::parameters &param) throw ()
- bool **saveTable** (const typename numericalFunctor< domainType, rangeType >::parameters &param) const throw ()
- bool **fixIndexVectors** (void)
- bool **verifyTableRange** (const rangeType &minInclusive, const rangeType &maxInclusive)
- long **numberOutOfRange** (const rangeType &minInclusive, const rangeType &maxInclusive)
- numericalFunctorLookupTable & **operator=** (const numericalFunctorLookupTable &other) throw ()
- void **copy** (const numericalFunctorLookupTable &other)
- std::string **uniqueFileName** (const typename numericalFunctor< domainType, rangeType >::parameters &param) const
- void **synchronizeUniqueID** (const typename numericalFunctor< domainType, rangeType >::parameters &param)
- void **setMaxNumberResident** (const long &n)
- const long & **maxNumberResident** (void)
- void **unloadAllResident** (void)

Protected Types

- typedef _STL_EXT_NAMESPACE_::hash_map< std::string, tableNode * > **hash_map**

Protected Member Functions

- std::string **tableListFileName** (const typename numericalFunctor< domainType, rangeType >::parameters &param) const
- std::string **fileContentsDescriptor** (const typename numericalFunctor< domainType, rangeType >::parameters &param) const
- void **unloadOldestResident** (void)
- void **addResidentTable** (const std::string &uniqueID)
- void **removeResidentTable** (const std::string &uniqueID) throw ()
- bool **current** (const std::string &uniqueID) const
- bool **swapResidentTables** (const std::string &uniqueID)

template<class domainType, class rangeType = domainType> class numerical_functor::numericalFunctorLookupTable< domainType, rangeType >

1.7.102 numerical_functor::numericalFunctorLookupTable< domainType, rangeType >::tableNode Class Reference

Public Member Functions

- tableNode & **operator=** (const tableNode &other)
- **tableNode** (const tableNode &other)
- **tableNode** (const long &nDim)

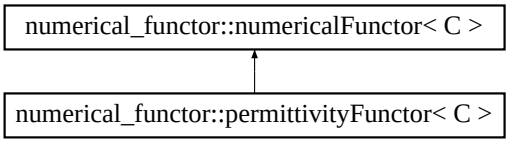
Public Attributes

- std::vector< rangeType > **table_**
- std::vector< std::vector< long > > * **pIndexVectors_**
- std::string **uniqueID_**
- long **timestamp_**

template<class domainType, class rangeType = domainType> class numerical_functor::numericalFunctorLookupTable< domainType, rangeType >::tableNode

1.7.103 numerical_functor::permittivityFunctor< C > Class Template Reference

Inheritance diagram for numerical_functor::permittivityFunctor< C >:



Classes

- class parameters

Public Types

- typedef numberTraits< C >::magnitudeType **R**
- typedef numberTraits< C >::integralType **Z**

Public Member Functions

- numericalFunctor< C > * **clone** (void) const
- const parameters & **getParameters** (void) const
- virtual bool **updateParameters** (void)
- bool **apply** (const R &E, C &epsilon, C &fError) const
- bool **apply** (const R &E, C &epsilon) const
- template<class V >
  bool **apply** (const V &vE, std::vector< C > &vEps, C &fError) const
- template<class V >
  bool **apply** (const V &vE, std::vector< C > &vEps) const
- bool **inDomain** (const R &arg) const throw ()

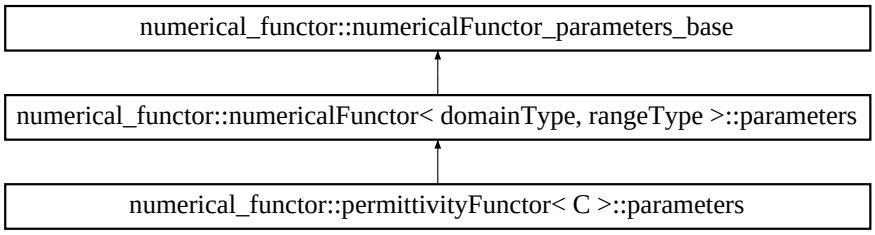
Static Public Member Functions

- static R **energyToWavelength** (const R &E, bool micron=false)
- static R **wavelengthToEnergy** (const R &lambda, bool micron=false)

template<class C> class numerical_functor::permittivityFunctor< C >

1.7.104 numerical_functor::permittivityFunctor< C >::parameters Class Reference

Inheritance diagram for numerical_functor::permittivityFunctor< C >::parameters:



Public Types

- typedef numericalFunctor< C >::parameters **base_class**

Public Member Functions

- bool **valid** (void) const
- parameters & **copy** (const parameters &other)
- virtual numericalFunctor< C >::parameters * **clone** (void) const
- **parameters** (const parameters &param)

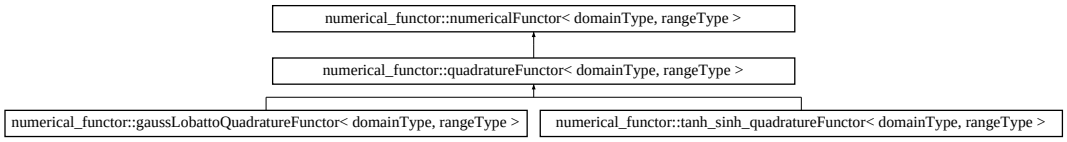
Public Attributes

- long **precision**
- permittivityData::material **eMaterial**
- R **A**
- R **d**
- bool **sizeCorrection**
- std::vector< C > **coeff**
- bool **refractiveIndex**

template<class C> class numerical_functor::permittivityFunctor< C >::parameters

1.7.105 numerical_functor::quadratureFunctor< domainType, rangeType > Class Template Reference

Inheritance diagram for numerical_functor::quadratureFunctor< domainType, rangeType >:



Classes

- class parameters

Public Types

- typedef number::numberTraits< domainType >::magnitudeType **R**

Public Member Functions

- **quadratureFunctor** (const quadratureFunctor &other) throw ()
- quadratureFunctor & **operator=** (const quadratureFunctor &other)
- quadratureFunctor & **copy** (const quadratureFunctor &other)
- numericalFunctor< domainType, rangeType > * **clone** (void) const

- virtual bool **setParameters** (const typename numericalFunctor< domainType, rangeType >::parameters &param)
- const parameters & **getParameters** (void) const
- virtual bool **apply** (const continuumFunctor< domainType, rangeType > &integrandFunctor, const domainType &intervalStart, const domainType &intervalEnd, rangeType &val, rangeType &fError) const
- bool **apply** (const continuumFunctor< domainType, rangeType > &integrandFunctor, const domainType &intervalStart, const domainType &intervalEnd, rangeType &val) const
- bool **apply** (const continuumFunctor< domainType, rangeType > &integrandFunctor, const std::vector< domainType > &vSubintervalPoints, rangeType &val, rangeType &fError) const
- bool **apply** (const continuumFunctor< domainType, rangeType > &integrandFunctor, const std::vector< domainType > &vSubintervalPoints, rangeType &val) const
- bool **inDomain** (const continuumFunctor< domainType, rangeType > &integrandFunctor, const long &arg) const
- bool **validSubintervalPoints** (const continuumFunctor< domainType, rangeType > &integrandFunctor, const std::vector< domainType > &vPoints) const
- bool **validInterval** (const continuumFunctor< domainType, rangeType > &integrandFunctor, const domainType &intervalStart, const domainType &intervalEnd) const
- const unsigned long & **functionCount** (void)
- void **clearRecursionCache** (void) const
- double **cacheUtilization** (void)

- unsigned long **cacheSize** (void)
- const rangeType & **lastEstimate** (void) const
- const unsigned long & **relTolAchievedCount** (void) const
- const unsigned long & **absTolAchievedCount** (void) const

Protected Attributes

- unsigned long **functionCount_**

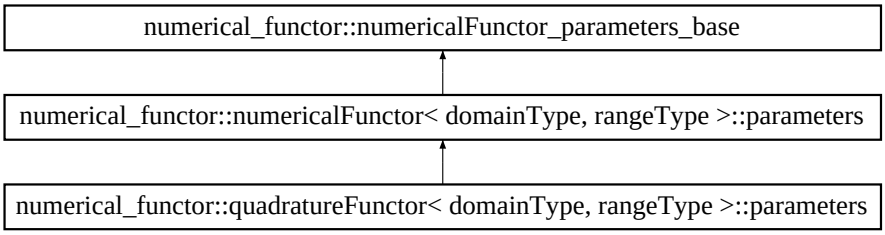
Friends

- class **continuumFunctor< domainType, rangeType >::quadratureCacheNode**

template<class domainType, class rangeType = domainType> class numerical_functor::quadratureFunctor< domainType, rangeType >

1.7.106 numerical_functor::quadratureFunctor< domainType, rangeType >::parameters Class Reference

Inheritance diagram for numerical_functor::quadratureFunctor< domainType, rangeType >::parameters:



Public Member Functions

- **parameters** (const parameters &param)
- parameters & **copy** (const parameters &other)
- numericalFunctor< domainType, rangeType >::parameters * **clone** (void) const
- bool **valid** (void) const
- virtual std::string **parentTypeName** (void) const

Public Attributes

- long **precision**
- numberTraits< rangeType >::magnitudeType **relativeTolerance**
- numberTraits< rangeType >::magnitudeType **absoluteTolerance**
- unsigned long **maxFunctionCount**
- bool **useRecursionCache**

template<class domainType, class rangeType = domainType> class numerical_functor::quadratureFunctor< domainType, rangeType >::parameters

1.7.107 numerical_functor::regionKet< T > Class Template Reference

Public Member Functions

- **regionKet** (const regionSelector< T > &region, const fieldKetBase< T > &ket)
- **regionKet** (const regionKet &other)

- regionKet & **operator=** (const regionKet &other)
- void **copy** (const regionKet &other)
- const regionSelector< T > & **region** (void) const throw ()
- regionSelector< T > & **region** (void) throw ()
- const fieldKetBase< T > & **ket** (void) const
- fieldKetBase< T > & **ket** (void)
- bool **writeBinary** (abstractCommHandle *fp) const
- bool **readBinary** (abstractCommHandle *fp)

Protected Member Functions

- void **freep** (void)

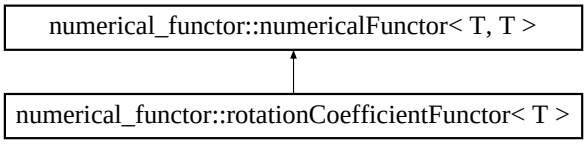
Protected Attributes

- regionSelector< T > * **pregion_**
- fieldKetBase< T > * **pket_**

template<class T> class numerical_functor::regionKet< T >

1.7.108 numerical_functor::rotationCoefficientFunctor< T > Class Template Reference

Inheritance diagram for numerical_functor::rotationCoefficientFunctor< T >:



Classes

- class parameters

Public Types

- typedef numberTraits< T >::magnitudeType **R**
- typedef size_t(* **indexFoop**)(long, long, size_t)
- typedef void(* **inverseIndexFoop**)(size_t, size_t, long &, long &)

Public Member Functions

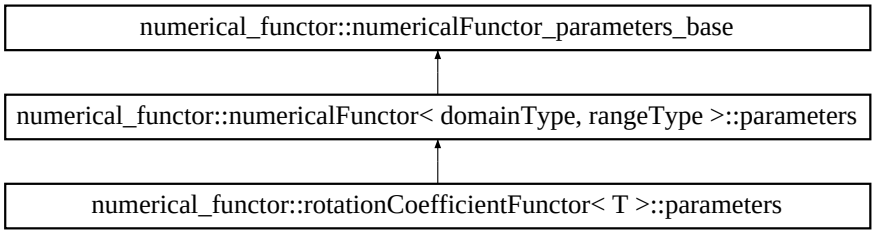
- **rotationCoefficientFunctor** (const rotationCoefficientFunctor &other)
- rotationCoefficientFunctor & **operator=** (const rotationCoefficientFunctor &other)
- rotationCoefficientFunctor & **copy** (const rotationCoefficientFunctor &other)
- numericalFunctor< T > * **clone** (void) const
- bool **setParameters** (const typename numericalFunctor< T, T >::parameters &param)
- const parameters & **getParameters** (void) const
- template<class MAT >
  bool **apply** (const R &alpha, const R &beta, const R &gamma, const MAT

&dest_) const throw ()

template<class T> class numerical_functor::rotationCoefficientFunctor< T >

1.7.109 numerical_functor::rotationCoefficientFunctor< T >::parameters Class Reference

Inheritance diagram for numerical_functor::rotationCoefficientFunctor< T >::parameters:



Public Types

- typedef numberTraits< T >::magnitudeType **R**

Public Member Functions

- parameters & **operator=** (const parameters &other)
- parameters & **copy** (const parameters &other)
- numericalFunctor< T >::parameters * **clone** (void) const
- bool **valid** (void) const
- **parameters** (const parameters &param)

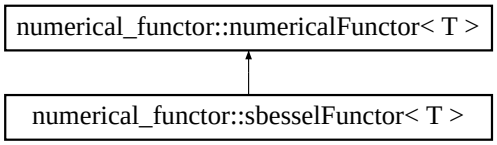
Public Attributes

- long **l_max**
- long **MIN_MULTIPOLE_ORDER**
- indexFoop **index**
- inverseIndexFoop **inverseIndex**

template<class T> class numerical_functor::rotationCoefficientFunctor< T >::parameters

1.7.110 numerical_functor::sbesselFunctor< T > Class Template Reference

Inheritance diagram for numerical_functor::sbesselFunctor< T >:



Classes

- class parameters

Public Types

- typedef numberTraits< T >::magnitudeType **magnitudeType**

Public Member Functions

- numericalFunctor< T > * **clone** (void) const

- const parameters & **getParameters** (void) const
- bool **apply** (const T &x, std::vector< T > &vjl, std::vector< T > &vDjl, std::vector< T > &vyl, std::vector< T > &vDyl, T &fError) const
- bool **apply** (const T &x, std::vector< T > &vjl, std::vector< T > &vDjl, std::vector< T > &vyl, std::vector< T > &vDyl) const
- bool **apply** (const T &x, std::vector< T > &vjl, std::vector< T > &vyl, T &fError) const
- bool **apply** (const T &x, std::vector< T > &vjl, std::vector< T > &vyl) const
- template<class V1 >
  bool **apply** (const T &x, const V1 &vjl_, T &fError) const
- template<class V1 >
  bool **apply** (const T &x, const V1 &vjl_) const
- bool **apply** (const T &x, std::vector< T > &vjl, T &fError) const
- bool **apply** (const T &x, std::vector< T > &vjl) const
- template<class V >
  bool **apply** (const V &vX, gmm::dense_matrix< T > &ajl, gmm::dense_-matrix< T > &aDjl, gmm::dense_matrix< T > &ayl, gmm::dense_matrix< T > &aDyl, T &fError) const
- template<class V >
  bool **apply** (const V &vX, gmm::dense_matrix< T > &ajl, gmm::dense_-matrix< T > &aDjl, gmm::dense_matrix< T > &ayl, gmm::dense_matrix< T > &aDyl) const
- template<class V >
  bool **apply** (const V &vX, gmm::dense_matrix< T > &ajl, gmm::dense_-matrix< T > &ayl, T &fError) const

- template<class V >

  bool **apply** (const V &vX, gmm::dense_matrix< T > &ajl, gmm::dense_-matrix< T > &ayl) const

Protected Member Functions

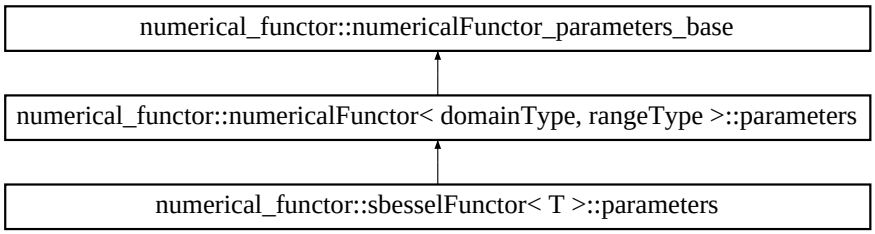
- template<class V1 , class V2 , class V3 , class V4 >

  bool **apply_** (const T &x, const V1 &vjl_, const V2 &vDjl_, const V3 &vyl_, const V4 &vDyl_, T &fError) const

template<class T> class numerical_functor::sbesselFunctor< T >

1.7.111 numerical_functor::sbesselFunctor< T >::parameters Class Reference

Inheritance diagram for numerical_functor::sbesselFunctor< T >::parameters:



Public Types

- enum **bessel_kind** {

  bessel, **bessel_first** = bessel, **neumann**, **bessel_second** = neumann,

hankel, **hankel_first** = hankel, **hankel_second** }

Public Member Functions

- **parameters** (const parameters &param)
- parameters & **copy** (const parameters &other)
- numericalFunctor< T >::parameters * **clone** (void) const
- bool **valid** (void) const

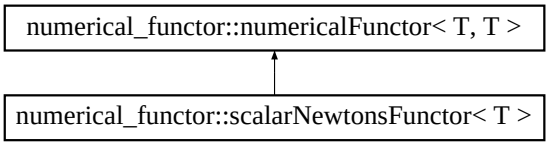
Public Attributes

- long **lMin**
- long **lMax**
- long **precision**
- long **nmaxIterations**
- T **minX**
- T **maxX**
- bessel_kind **eKind**
- bool **mieDerivForm**

template<class T> class numerical_functor::sbesselFunctor< T >::parameters

1.7.112 numerical_functor::scalarNewtonsFunctor< T > Class Template Reference

Inheritance diagram for numerical_functor::scalarNewtonsFunctor< T >:



Classes

- struct iterationHistoryStruct
- class parameters

Public Types

- typedef numberTraits< T >::magnitudeType **R**
- typedef T(∗ **minimizeFunction**)(const T &arg, void ∗parms)

Public Member Functions

- **scalarNewtonsFunctor** (const scalarNewtonsFunctor &other)
- scalarNewtonsFunctor & **operator=** (const scalarNewtonsFunctor &other)
- scalarNewtonsFunctor & **copy** (const scalarNewtonsFunctor &other)
- numericalFunctor< T > ∗ **clone** (void) const
- bool **setParameters** (const typename numericalFunctor< T, T >::parameters &param)
- const parameters & **getParameters** (void) const
- bool **apply** (minimizeFunction func, const T &x0, T &x, void ∗pparms=NULL) const throw ()
- const std::vector< iterationHistoryStruct > & **history** (void) const

template<class T> class numerical_functor::scalarNewtonsFunctor< T >

1.7.113 numerical_functor::scalarNewtonsFunctor< T >::iterationHistoryStruct Struct Reference

Public Types

- typedef numberTraits< T >::magnitudeType **R**

Public Member Functions

- void **write** (std::ostream &os) const
- **iterationHistoryStruct** (const R &res, const T &iter, size_t N_steps)

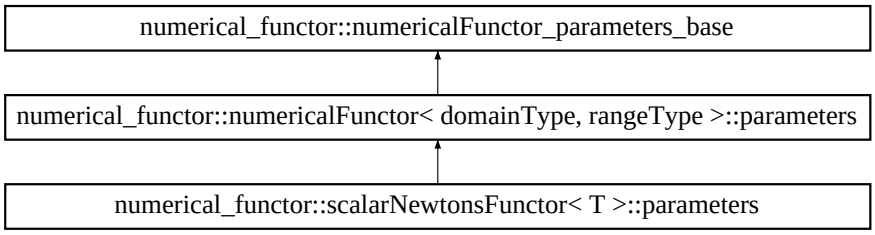
Public Attributes

- R **residual**
- T **iterate**
- size_t **N_innerSteps**

template<class T> struct numerical_functor::scalarNewtonsFunctor< T >::iterationHistoryStruct

1.7.114 numerical_functor::scalarNewtonsFunctor< T >::parameters Class Reference

Inheritance diagram for numerical_functor::scalarNewtonsFunctor< T >::parameters:



Public Types

- typedef numberTraits< T >::magnitudeType **R**

Public Member Functions

- parameters & **operator=** (const parameters &other)
- parameters & **copy** (const parameters &other)
- numericalFunctor< T >::parameters * **clone** (void) const
- bool **valid** (void) const
- **parameters** (const parameters &param)

Public Attributes

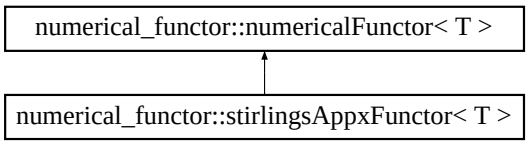
- R **absoluteTolerance**
- R **relativeTolerance**
- size_t **N_maxIterations**
- size_t **N_maxInnerIterations**
- R **derivativeStep**
- R **smallDerivativeLimit**
- R **lineSearchContractionEfficiencyFactor**
- R **lineSearchCutoff**

- R **minRealArg**
- R **maxRealArg**
- R **minImagArg**
- R **maxImagArg**
- bool **bHistory**

template<class T> class numerical_functor::scalarNewtonsFunctor< T >::parameters

1.7.115 numerical_functor::stirlingsAppxFunctor< T > Class Template Reference

Inheritance diagram for numerical_functor::stirlingsAppxFunctor< T >:



Classes

- class parameters

Public Member Functions

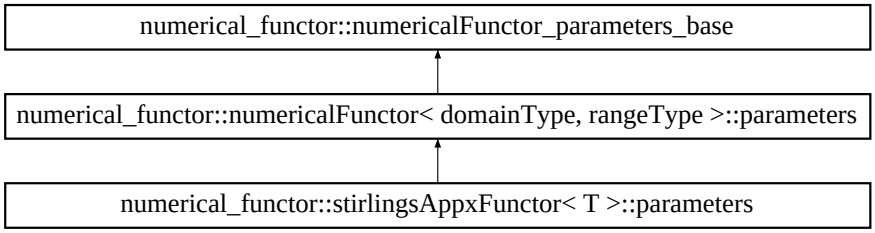
- numericalFunctor< T > * **clone** (void) const
- const parameters & **getParameters** (void)
- bool **apply** (const T &arg, T &val, T &fError)

- bool **apply** (const T &arg, T &val)

template<class T> class numerical_functor::stirlingsAppxFunctor< T >

1.7.116 numerical_functor::stirlingsAppxFunctor< T >::parameters Class Reference

Inheritance diagram for numerical_functor::stirlingsAppxFunctor< T >::parameters:



Public Member Functions

- **parameters** (const parameters &param)
- parameters & **copy** (const parameters &other)
- numericalFunctor< T >::parameters * **clone** (void) const
- bool **valid** (void) const

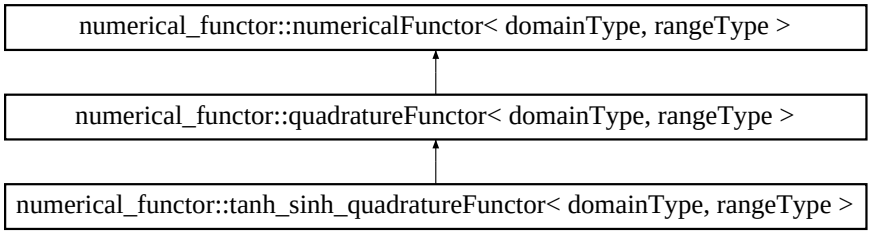
Public Attributes

- long **precision**
- T **minX**
- T **maxX**

template<class T> class numerical_functor::stirlingsAppxFunctor< T >::parameters

1.7.117 numerical_functor::tanh_sinh_quadratureFunctor< domainType, rangeType > Class Template Reference

Inheritance diagram for numerical_functor::tanh_sinh_quadratureFunctor< domainType, rangeType >:



Public Types

- typedef number::numberTraits< domainType >::magnitudeType **R**

Public Member Functions

- numericalFunctor< domainType, rangeType > * **clone** (void) const
- virtual bool **apply** (const continuumFunctor< domainType, rangeType > &integrandFunctor, const domainType &intervalStart, const domainType &intervalEnd, rangeType &val, rangeType &fError) const

template<class domainType, class rangeType = domainType> class numerical_functor::tanh_sinh_quadratureFunctor< domainType, rangeType >

1.7.118 numerical_functor::timestamp Class Reference

Static Public Member Functions

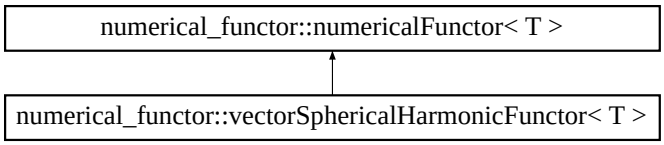
- static unsigned long **uniqueTimestamp** (void)
- static unsigned long & **current** (void)
- static bool **older** (const unsigned long &timestampA, const unsigned long &timestampB) throw ()

Static Protected Member Functions

- static unsigned long **age** (const unsigned long &testTimestamp)

1.7.119 numerical_functor::vectorSphericalHarmonicFunctor< T > Class Template Reference

Inheritance diagram for numerical_functor::vectorSphericalHarmonicFunctor< T >:



Classes

- class parameters
- class **workspace**

Public Types

- enum **VSH_KIND** { **VSH_P**, **VSH_B**, **VSH_C** }

Public Member Functions

- void **copy** (const vectorSphericalHarmonicFunctor< T > &other)
- **vectorSphericalHarmonicFunctor** (const vectorSphericalHarmonicFunctor< T > &other)
- numericalFunctor< T > * **clone** (void) const
- const parameters & **getParameters** (void) const
- virtual bool **updateParameters** (void)
- bool **apply** (const T &t, const T &p, gmm::dense_matrix< T > &B_lm, gmm::dense_matrix< T > &C_lm, gmm::dense_matrix< T > &P_lm, T &fError) const
- bool **apply** (const T &t, const T &p, gmm::dense_matrix< T > &B_lm, gmm::dense_matrix< T > &C_lm, gmm::dense_matrix< T > &P_lm) const

- bool **apply** (const T &t, const T &p, std::vector< T > &B_lm_r, std::vector< T > &B_lm_t, std::vector< T > &B_lm_p, std::vector< T > &C_lm_r, std::vector< T > &C_lm_t, std::vector< T > &C_lm_p, std::vector< T > &P_lm_r, std::vector< T > &P_lm_t, std::vector< T > &P_lm_p, T &fError) const
- bool **apply** (const T &t, const T &p, std::vector< T > &B_lm_r, std::vector< T > &B_lm_t, std::vector< T > &B_lm_p, std::vector< T > &C_lm_r, std::vector< T > &C_lm_t, std::vector< T > &C_lm_p, std::vector< T > &P_lm_r, std::vector< T > &P_lm_t, std::vector< T > &P_lm_p) const

- bool **apply** (const std::vector< T > &vT, const std::vector< T > &vP, gmm::dense_matrix< T > &B_lm_r, gmm::dense_matrix< T > &B_lm_t, gmm::dense_matrix< T > &B_lm_p, gmm::dense_matrix< T > &C_lm_r, gmm::dense_matrix< T > &C_lm_t, gmm::dense_matrix< T > &C_lm_p, gmm::dense_matrix< T > &P_lm_r, gmm::dense_matrix< T > &P_lm_t, gmm::dense_matrix< T > &P_lm_p, T &fError) const
- bool **apply** (const std::vector< T > &vT, const std::vector< T > &vP, gmm::dense_matrix< T > &B_lm_r, gmm::dense_matrix< T > &B_lm_t, gmm::dense_matrix< T > &B_lm_p, gmm::dense_matrix< T > &C_lm_r, gmm::dense_matrix< T > &C_lm_t, gmm::dense_matrix< T > &C_lm_p, gmm::dense_matrix< T > &P_lm_r, gmm::dense_matrix< T > &P_lm_t, gmm::dense_matrix< T > &P_lm_p) const
- bool **apply** (const T &t, const T &p, VSH_KIND eVSH, gmm::dense_-matrix< T > &U_lm, T &fError) const
- bool **apply** (const T &t, const T &p, VSH_KIND eVSH, gmm::dense_-matrix< T > &U_lm) const
- bool **apply** (const T &t, const T &p, VSH_KIND eVSH, std::vector< T > &U_lm_r, std::vector< T > &U_lm_t, std::vector< T > &U_lm_p, T &fEr-ror) const
- bool **apply** (const T &t, const T &p, VSH_KIND eVSH, std::vector< T > &U_lm_r, std::vector< T > &U_lm_t, std::vector< T > &U_lm_p) const
- bool **apply** (const std::vector< T > &vT, const std::vector< T > &vP, VSH_KIND eVSH, gmm::dense_matrix< T > &U_lm_r, gmm::dense_-matrix< T > &U_lm_t, gmm::dense_matrix< T > &U_lm_p, T &fError) const

- bool **apply** (const std::vector< T > &vT, const std::vector< T > &vP, VSH_KIND eVSH, gmm::dense_matrix< T > &U_lm_r, gmm::dense_matrix< T > &U_lm_t, gmm::dense_matrix< T > &U_lm_p) const
- bool **B** (const T &t, const T &p, gmm::dense_matrix< T > &B_lm, T &fError) const
- bool **B** (const T &t, const T &p, gmm::dense_matrix< T > &B_lm) const
- bool **B** (const T &t, const T &p, std::vector< T > &B_lm_r, std::vector< T > &B_lm_t, std::vector< T > &B_lm_p, T &fError) const
- bool **B** (const T &t, const T &p, std::vector< T > &B_lm_r, std::vector< T > &B_lm_t, std::vector< T > &B_lm_p) const
- bool **B** (const std::vector< T > &vT, const std::vector< T > &vP, gmm::dense_matrix< T > &B_lm_r, gmm::dense_matrix< T > &B_lm_t, gmm::dense_matrix< T > &B_lm_p, T &fError) const
- bool **B** (const std::vector< T > &vT, const std::vector< T > &vP, gmm::dense_matrix< T > &B_lm_r, gmm::dense_matrix< T > &B_lm_t, gmm::dense_matrix< T > &B_lm_p) const
- bool **C** (const T &t, const T &p, gmm::dense_matrix< T > &C_lm, T &fError) const
- bool **C** (const T &t, const T &p, gmm::dense_matrix< T > &C_lm) const
- bool **C** (const T &t, const T &p, std::vector< T > &C_lm_r, std::vector< T > &C_lm_t, std::vector< T > &C_lm_p, T &fError) const
- bool **C** (const T &t, const T &p, std::vector< T > &C_lm_r, std::vector< T > &C_lm_t, std::vector< T > &C_lm_p) const
- bool **C** (const std::vector< T > &vT, const std::vector< T > &vP, gmm::dense_matrix< T > &C_lm_r, gmm::dense_matrix< T > &C_lm_t, gmm::dense_matrix< T > &C_lm_p, T &fError) const

- bool **C** (const std::vector< T > &vT, const std::vector< T > &vP, gmm::dense_matrix< T > &C_lm_r, gmm::dense_matrix< T > &C_lm_t, gmm::dense_matrix< T > &C_lm_p) const
- bool **P** (const T &t, const T &p, gmm::dense_matrix< T > &P_lm, T &fError) const
- bool **P** (const T &t, const T &p, gmm::dense_matrix< T > &P_lm) const
- bool **P** (const T &t, const T &p, std::vector< T > &P_lm_r, std::vector< T > &P_lm_t, std::vector< T > &P_lm_p, T &fError) const
- bool **P** (const T &t, const T &p, std::vector< T > &P_lm_r, std::vector< T > &P_lm_t, std::vector< T > &P_lm_p) const
- bool **P** (const std::vector< T > &vT, const std::vector< T > &vP, gmm::dense_matrix< T > &P_lm_r, gmm::dense_matrix< T > &P_lm_t, gmm::dense_matrix< T > &P_lm_p, T &fError) const
- bool **P** (const std::vector< T > &vT, const std::vector< T > &vP, gmm::dense_matrix< T > &P_lm_r, gmm::dense_matrix< T > &P_lm_t, gmm::dense_matrix< T > &P_lm_p) const

Protected Member Functions

- template<class V >

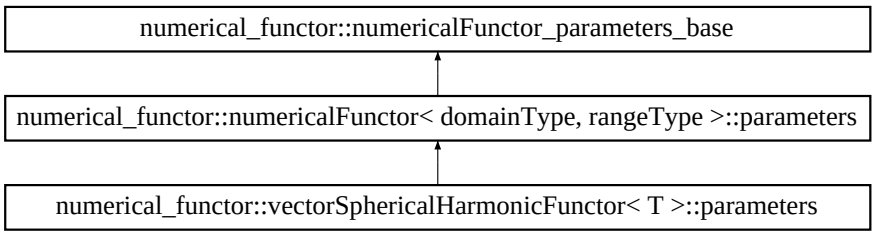
  bool **apply_** (const T &t, const T &p, const V &vB_lm_r_, const V &vB_lm_t_, const V &vB_lm_p_, const V &vC_lm_r_, const V &vC_lm_t_, const V &vC_lm_p_, const V &vP_lm_r_, const V &vP_lm_t_, const V &vP_lm_p_, T &fError) const
- template<class V >

  bool **P_B_C_** (const T &t, const T &p, VSH_KIND eVSH, const V &vB_lm_r_, const V &vB_lm_t_, const V &vB_lm_p_, T &fError) const

template<class T> class numerical_functor::vectorSphericalHarmonicFunctor< T >

1.7.120 numerical_functor::vectorSphericalHarmonicFunctor< T >::parameters Class Reference

Inheritance diagram for numerical_functor::vectorSphericalHarmonicFunctor< T >::parameters:



Public Member Functions

- **parameters** (const parameters &param)
- parameters & **copy** (const parameters &other)
- numericalFunctor< T >::parameters * **clone** (void) const
- bool **valid** (void) const

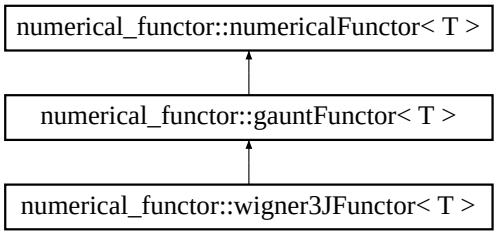
Public Attributes

- long **m**
- long **lMax**
- long **precision**
- bool **cosTheta**

template<class T> class numerical_functor::vectorSphericalHarmonicFunctor< T >::parameters

1.7.121 numerical_functor::wigner3JFunctor< T > Class Template Reference

Inheritance diagram for numerical_functor::wigner3JFunctor< T >:



Classes

- class parameters

Public Types

- typedef numberTraits< T >::integralType **Z**

Public Member Functions

- numericalFunctor< T > * **clone** (void) const
- const parameters & **getParameters** (void) const
- virtual bool **updateParameters** (void)
- bool **apply** (const long &l1, const long &l2, const long &l3, const long &m1, const long &m2, const long &m3, T &val, T &fError, bool

bForceNoLookupTable=false) const throw ()

- bool **apply** (const long &l1, const long &l2, const long &l3, const long &m1, const long &m2, const long &m3, T &val) const throw ()
- T **operator()** (const long &l1, const long &l2, const long &l3, const long &m1, const long &m2, const long &m3) const throw ()
- bool **apply** (const multipoleSextuple &s6, T &val, T &fError, bool bForceNoLookupTable=false) const throw ()
- bool **apply** (const multipoleSextuple &s6, T &val) const throw ()

Static Public Member Functions

- static bool **isTrivialZero** (const multipoleSextuple &s6Arg)
- static bool **isTrivialZero** (const long &l1, const long &l2, const long &l3, const long &m1, const long &m2, const long &m3)

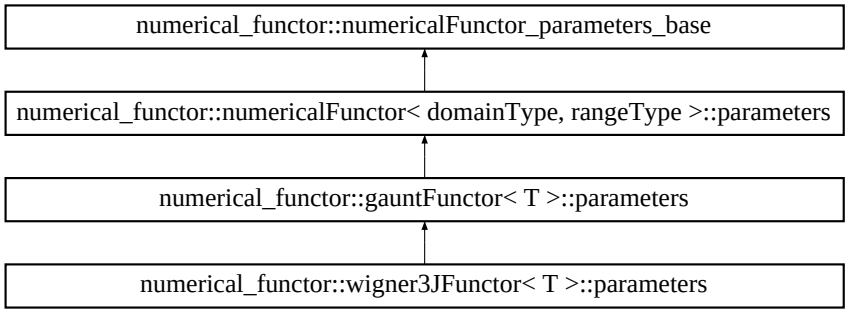
Protected Member Functions

- bool **apply_** (const multipoleSextuple &s6ArgStandardForm, T &val, T &fError) const throw ()
- void **permuteCorrection** (const multipoleSextuple &s6Arg, typename multipoleSextuple::permutation::permutation_Kind requiredPermute, T &val) const
- void **clebschGordanNormalize** (const multipoleSextuple &s6Arg, T &val) const
- virtual bool **createLookupTable** (void)

template<class T> class numerical_functor::wigner3JFunctor< T >

1.7.122 numerical_functor::wigner3JFunctor< T >::parameters Class Reference

Inheritance diagram for numerical_functor::wigner3JFunctor< T >::parameters:



Public Member Functions

- **parameters** (const parameters &param)
- parameters & **copy** (const parameters &other)
- numericalFunctor< T >::parameters * **clone** (void) const
- bool **valid** (void) const
- virtual std::string **parentTypeName** (void) const
- virtual std::string **lookupTableAppendString** (void) const
- virtual bool **evenLSumRule** (void) const

Public Attributes

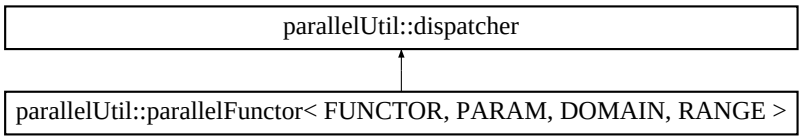
- bool **clebschGordanNormalization**

template<class T> class numerical_functor::wigner3JFunctor< T >::parameters

1.7.123 parallelUtil::dispatcher Class Reference

dispatcher/receiver task-based MPI implementation.

Inheritance diagram for parallelUtil::dispatcher:



Classes

- class parameters
- class workspace

Public Types

- enum **MESSAGE_KIND** {

  NO_MESSAGE = 0, **TASK_ASSIGN** = 10, **TASK_COMPLETE** = 20, **TASK_ERROR** = 30,

  INIT_SHARE = 40, **FINISH_SHARE** = 50, **COMMAND_FINISH_-SHARE** = 60, **COMMAND_RECEIVER_EXIT** = 70 }

Public Member Functions

- bool **setParameters** (const parameters &param)
- const parameters & **getParameters** (void) const
- void **setStatusString** (const std::string &sMsg) const
- const std::string & **getStatusString** (void) const
- const MPI::Intracomm & **receiversComm** (void) const
- void **receiveErrorInfo** (size_t np) const throw ()
- void **commandReceiver** (size_t np, MESSAGE_KIND tag) const throw ()
- void **sendErrorInfo** (const std::string &sMsg, size_t np) const throw ()
- void **receiveTag** (size_t np, int &tag) const throw ()
- void **work** (workspace &ws) const throw ()

Static Public Member Functions

- static void **write** (std::ostream &os, MESSAGE_KIND e) throw ()
- static void **dispatcherThreadFunc** (const dispatcher &owner, workspace &ws) throw ()
- static void **receiverThreadFunc** (const dispatcher &owner, workspace &ws) throw ()

1.7.123.1 Detailed Description

dispatcher/receiver task-based MPI implementation.

1.7.124 parallelUtil::dispatcher::parameters Class Reference

Public Member Functions

- const MPI::Comm & **comm** (void) const
- void **setComm** (const MPI::Comm &newComm)

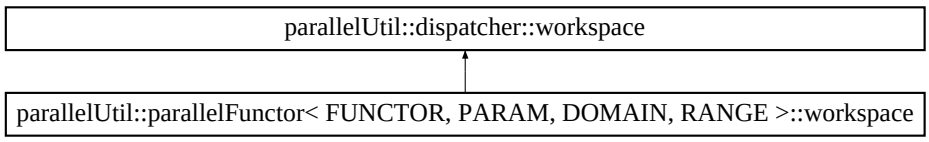
- parameters & **operator=** (const parameters &other)

Public Attributes

- bool **lastDispatcher**
- bool **zeroIsSpecial**
- bool **initShare**
- bool **finishShareOut**
- bool **finishShareAcross**
- size_t **threadsPerRank**
- size_t **bufSize**

1.7.125 parallelUtil::dispatcher::workspace Class Reference

Inheritance diagram for parallelUtil::dispatcher::workspace:



Classes

- class result
- class task

Public Member Functions

- virtual task ∗ **newTask** (void) const =0
- virtual result ∗ **newResult** (void) const =0
- virtual bool **getTask** (task &t)=0 throw ()
- virtual void **collectTaskResult** (const result &r)=0 throw ()
- virtual bool **allComplete** (void) const
- virtual void **finish** (void)
- virtual void **sendInitShare** (abstractCommHandle ∗h) const throw ()
- virtual void **sendFinishShare** (abstractCommHandle ∗h) const throw ()
- const size_t & **tasksAssigned** (void) const
- const size_t & **tasksCollected** (void) const
- virtual void **executeTask** (const task &t, result &r)=0 throw ()
- virtual void **receiveInitShare** (abstractCommHandle ∗h) throw ()
- virtual void **receiveFinishShare** (abstractCommHandle ∗h) throw ()
- virtual void **init** (bool isDispatcher) throw ()
- virtual void **finish** (bool isDispatcher) throw ()
- virtual workspace ∗ **clone** (void)=0 throw ()

Protected Member Functions

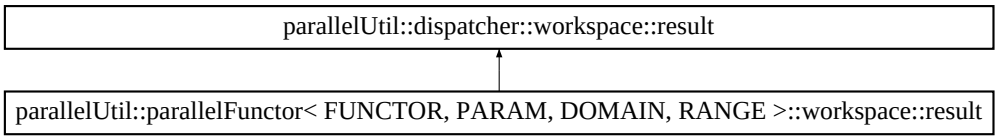
- const dispatcher & **owner** (void) const throw ()

Friends

- class **dispatcher**

1.7.126 parallelUtil::dispatcher::workspace::result Class Reference

Inheritance diagram for parallelUtil::dispatcher::workspace::result:

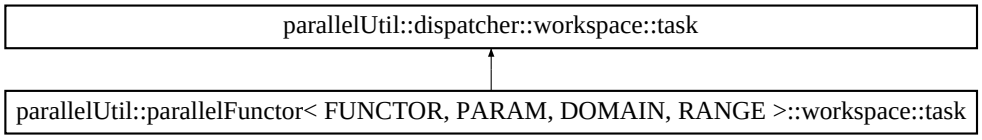


Public Member Functions

- virtual bool **writeBinary** (abstractCommHandle ∗h) const =0 throw ()
- virtual bool **readBinary** (abstractCommHandle ∗h)=0 throw ()
- virtual void **write** (std::ostream &os) const =0 throw ()

1.7.127 parallelUtil::dispatcher::workspace::task Class Reference

Inheritance diagram for parallelUtil::dispatcher::workspace::task:

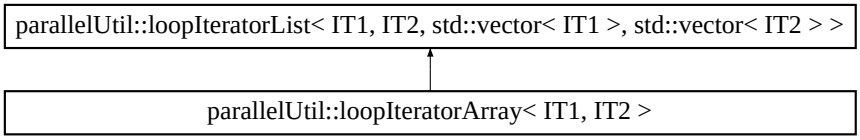


Public Member Functions

- virtual bool **writeBinary** (abstractCommHandle ∗h) const =0 throw ()
- virtual bool **readBinary** (abstractCommHandle ∗h)=0 throw ()
- virtual void **write** (std::ostream &os) const =0 throw ()

1.7.128 parallelUtil::loopIteratorArray< IT1, IT2 > Class Template Reference

Inheritance diagram for parallelUtil::loopIteratorArray< IT1, IT2 >:



Public Types

- typedef std::vector< IT1 > **U1**
- typedef std::vector< IT2 > **U2**
- typedef loopIteratorList< IT1, IT2, U1, U2 > **base_class**

Public Member Functions

- const IT1 & **it** (int n) const
- const IT2 & **it2** (int n) const
- const IT1 & **operator[ ]** (size_t n) const
- loopIteratorArray & **operator=** (const loopIteratorArray &other)
- virtual base_class * **clone** (void) const
- **loopIteratorArray** (const IT1 &it1Start, const IT1 &it1End, const IT2 &it2Start)
- **loopIteratorArray** (const IT1 &it1Start, const IT1 &it1End)
- **loopIteratorArray** (const loopIteratorArray &other)

Protected Member Functions

- void **copy** (const loopIteratorArray &other)

template<class IT1, class IT2 = IT1> class parallelUtil::loopIteratorArray< IT1, IT2 >

1.7.129 parallelUtil::loopIteratorList< IT1, IT2, U1, U2 > Class Template Reference

Public Types

- typedef U1::iterator **iterator**
- typedef U1::const_iterator **const_iterator**
- typedef U1::reverse_iterator **reverse_iterator**
- typedef U1::const_reverse_iterator **const_reverse_iterator**
- typedef U2::iterator **iterator2**
- typedef U2::const_iterator **const_iterator2**
- typedef U2::reverse_iterator **reverse_iterator2**
- typedef U2::const_reverse_iterator **const_reverse_iterator2**

Public Member Functions

- iterator **begin** (void)
- const_iterator **begin** (void) const
- reverse_iterator **rbegin** (void)
- const_reverse_iterator **rbegin** (void) const
- iterator **end** (void)

- const_iterator **end** (void) const
- reverse_iterator **rend** (void)
- const_reverse_iterator **rend** (void) const
- iterator **erase** (iterator pos)
- bool **empty** (void) const
- size_t **size** (void) const
- void **clear** (void)
- iterator2 **begin2** (void)
- const_iterator2 **begin2** (void) const
- reverse_iterator2 **rbegin2** (void)
- const_reverse_iterator2 **rbegin2** (void) const
- iterator2 **end2** (void)
- const_iterator2 **end2** (void) const
- reverse_iterator2 **rend2** (void)
- const_reverse_iterator2 **rend2** (void) const
- iterator **erase2** (iterator2 pos)
- bool **empty2** (void) const
- size_t **size2** (void) const
- const IT1 & **operator[ ]** (size_t n) const
- loopIteratorList & **operator=** (const loopIteratorList &other)
- loopIteratorList **split** (size_t n_split, size_t N_splits, bool comb=false) const

- void **split** (size_t n_split, size_t N_splits, bool comb, loopIteratorList &val) const

- loopIteratorList * **cloneSplit** (size_t n_split, size_t N_splits, bool comb=false) const
- virtual loopIteratorList * **clone** (void) const
- void **write** (std::ostream &os) const
- **loopIteratorList** (const IT1 &it1Start, const IT1 &it1End, const IT2 &it2Start)
- **loopIteratorList** (const IT1 &it1Start, const IT1 &it1End)
- **loopIteratorList** (const loopIteratorList &other)

Protected Member Functions

- void **copy** (const loopIteratorList &other)

Protected Attributes

- U1 **uIT1_**
- U2 **uIT2_**

template<class IT1, class IT2, class U1, class U2> class parallelUtil::loopIteratorList< IT1, IT2, U1, U2 >

1.7.130 parallelUtil::monolithic_dispatcher Class Reference

Single-task dispatcher/receiver MPI implementation.

Classes

- class parameters

- class workspace

Public Types

- enum **MESSAGE_KIND** {

  NO_MESSAGE = 0, **TASK_ERROR** = 30, **INIT_SHARE** = 40, **FINISH_-SHARE** = 50,

  COMMAND_FINISH_SHARE = 60, **COMMAND_RECEIVER_EXIT** = 70 }

Public Member Functions

- bool **setParameters** (const parameters &param)
- const parameters & **getParameters** (void) const
- void **setStatusString** (const std::string &sMsg) const
- const std::string & **getStatusString** (void) const
- const MPI::Intracomm & **receiversComm** (void) const
- void **receiveErrorInfo** (size_t np) const throw ()
- void **commandReceiver** (size_t np, MESSAGE_KIND tag) const throw ()
- void **sendErrorInfo** (const std::string &sMsg, size_t np) const throw ()
- void **receiveTag** (size_t np, int &tag) const throw ()
- void **work** (workspace &ws) const throw ()

Static Public Member Functions

- static void **write** (std::ostream &os, MESSAGE_KIND e) throw ()
- static void **dispatcherThreadFunc** (const monolithic_dispatcher &owner, workspace &ws) throw ()

- static void **receiverThreadFunc** (const monolithic_dispatcher &owner, workspace &ws) throw ()

1.7.130.1 Detailed Description

Single-task dispatcher/receiver MPI implementation. This class provides a single-task interface to complements the multiple-task dispatcher/receiver MPI implementation; in combination, these classes provide a unified interface to MPI.

1.7.131 parallelUtil::monolithic_dispatcher::parameters Class Reference

Public Member Functions

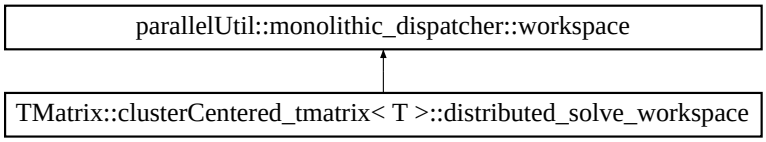
- const MPI::Comm & **comm** (void) const
- void **setComm** (const MPI::Comm &newComm)
- parameters & **operator=** (const parameters &other)

Public Attributes

- bool **lastDispatcher**
- bool **initShare**
- bool **finishShareOut**
- bool **finishShareAcross**
- size_t **threadsPerRank**
- size_t **bufSize**

1.7.132 parallelUtil::monolithic_dispatcher::workspace Class Reference

Inheritance diagram for parallelUtil::monolithic_dispatcher::workspace:



Public Member Functions

- virtual void **monolithicMPIBlock** (void)=0 throw ()
- virtual void **finish** (void)
- virtual void **sendInitShare** (abstractCommHandle ∗h) const throw ()
- virtual void **sendFinishShare** (abstractCommHandle ∗h) const throw ()
- virtual void **receiveInitShare** (abstractCommHandle ∗h) throw ()
- virtual void **receiveFinishShare** (abstractCommHandle ∗h) throw ()
- virtual void **init** (bool isDispatcher) throw ()
- virtual void **finish** (bool isDispatcher) throw ()
- virtual workspace ∗ **clone** (void)=0 throw ()

Protected Member Functions

- const monolithic_dispatcher & **owner** (void) const throw ()

Friends

- class **monolithic_dispatcher**

1.7.133 parallelUtil::multiMutex Class Reference

Classes

- struct **namedLock**

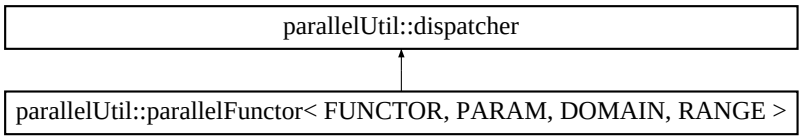
Public Member Functions

- void **reset** (void)
- size_t **lock** (unsigned long key) throw (std::string)
- bool **trylock** (unsigned long key, size_t &nLock) throw (std::string)
- template<class K >
  size_t **lock** (const K &k) throw (std::string)
- template<class K >
  bool **trylock** (const K &k, size_t &nLock) throw (std::string)
- void **unlock** (size_t nLock) throw (std::string)
- void **allocLocks** (size_t N_locks) throw (std::string)
- size_t **N_locks** (void)
- **multiMutex** (size_t N_locks=1)

1.7.134 parallelUtil::parallelFunctor< FUNCTOR, PARAM, DOMAIN, RANGE > Class Template Reference

generic functor to MPI-task mapping, using no shared workspace.

Inheritance diagram for parallelUtil::parallelFunctor< FUNCTOR, PARAM, DOMAIN, RANGE >:



Classes

- class workspace

  Derived from parallelUtil::dispatcher::workspace Not used externally.

Public Types

- typedef dispatcher **base_class**
- typedef FUNCTOR **functor**
- typedef PARAM **parameters**
- typedef DOMAIN **domain**
- typedef RANGE **range**

Public Member Functions

- bool **setParameters** (const std::vector< parameters > &param)
- const std::vector< parameters > & **getParameters** (void) const
- void **setStatusString** (const std::string &msg) const
- const std::string & **getStatusString** (void) const
- void **set_functor_status** (bool flag) const
- bool **functor_status** (void) const
- bool set_dispatcher_parameters (const typename base_class::parameters &param)

Initialize parameters for dispatcher associated with this parallelFunctor. This action must be duplicated by all ranks. At present, the only supported modifiable attributes are: "lastDispatcher", "zeroIsSpecial", "finishShareOut", "threadsPerRank", and "bufSize"; values set for other attributes will be ignored.

- const base_class::parameters & **get_dispatcher_parameters** (void) const
- bool **apply** (const std::vector< domain > &arg, std::vector< range > &val) throw (std::string)

Protected Member Functions

- bool **setParameters** (const typename base_class::parameters &param)

1.7.134.1 Detailed Description

template<class FUNCTOR, class PARAM, class DOMAIN, class RANGE> class parallelUtil::parallelFunctor< FUNCTOR, PARAM, DOMAIN, RANGE >

generic functor to MPI-task mapping, using no shared workspace.

Template Parameters

FUNCTOR functor class to be mapped to MPI

PARAM associated parameters class

DOMAIN argument class

RANGE return-value class

FUNCTOR shall provide the following methods:

- bool setParameters(const PARAM&)
- bool apply(const DOMAIN&, RANGE&)

- const std::string&getStatusString(void)const

input:

- std::vector<PARAM> available at rank-0 (contains exactly one, or number of values to match argument vector)
- std::vector<DOMAIN> available at rank-0

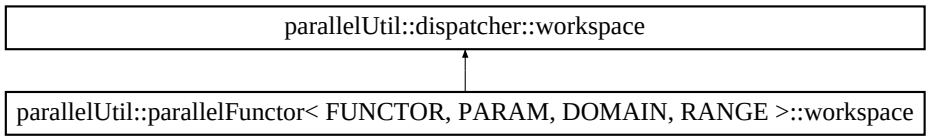
returns: std::vector<RANGE> available at rank-0 (and optionally at all ranks)

usage: return value vector does not need to be pre-allocated.

1.7.135 parallelUtil::parallelFunctor< FUNCTOR, PARAM, DOMAIN, RANGE >::workspace Class Reference

Derived from parallelUtil::dispatcher::workspace Not used externally.

Inheritance diagram for parallelUtil::parallelFunctor< FUNCTOR, PARAM, DOMAIN, RANGE >::workspace:



Classes

- class result
- class task

Public Types

- typedef dispatcher::workspace **base_class**

Public Member Functions

- virtual base_class::task * **newTask** (void) const
- virtual base_class::result * **newResult** (void) const
- const parallelFunctor< FUNCTOR, PARAM, DOMAIN, RANGE > & **owner** (void) const
- virtual bool **getTask** (base_class::task &t) throw ()
- virtual void **collectTaskResult** (const base_class::result &r) throw ()
- virtual void **sendFinishShare** (abstractCommHandle *h) const throw ()
- virtual void **executeTask** (const base_class::task &t, base_class::result &r) throw ()
- virtual void **receiveFinishShare** (abstractCommHandle *h) throw ()
- virtual void **init** (bool isDispatcher) throw ()
- virtual base_class::workspace * **clone** (void) throw ()
- **workspace** (const workspace &other, bool own_data=false)
- **workspace** (const parallelFunctor< FUNCTOR, PARAM, DOMAIN, RANGE > *powner, const std::vector< DOMAIN > &v_domain, std::vector< RANGE > &v_range)

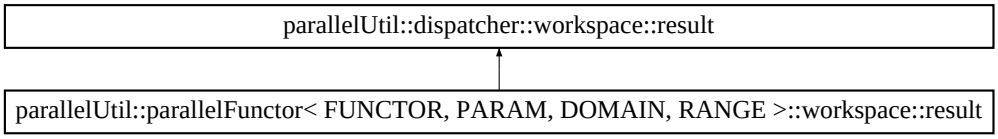
1.7.135.1 Detailed Description

template<class FUNCTOR, class PARAM, class DOMAIN, class RANGE> class parallelUtil::parallelFunctor< FUNCTOR, PARAM, DOMAIN, RANGE >::workspace

Derived from parallelUtil::dispatcher::workspace Not used externally.

1.7.136 parallelUtil::parallelFunctor< FUNCTOR, PARAM, DOMAIN, RANGE >::workspace::result Class Reference

Inheritance diagram for parallelUtil::parallelFunctor< FUNCTOR, PARAM, DOMAIN, RANGE >::workspace::result:



Public Member Functions

- virtual bool **writeBinary** (abstractCommHandle ∗h) const throw ()
- virtual bool **readBinary** (abstractCommHandle ∗h) throw ()
- virtual void **write** (std::ostream &os) const throw ()

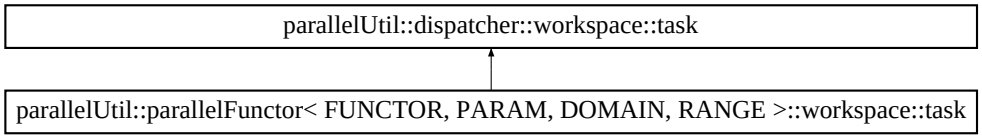
Public Attributes

- size_t **nresult**
- bool **valid**
- range **value**
- std::string **status_msg**

template<class FUNCTOR, class PARAM, class DOMAIN, class RANGE> class parallelUtil::parallelFunctor< FUNCTOR, PARAM, DOMAIN, RANGE >::workspace::result

1.7.137 parallelUtil::parallelFunctor< FUNCTOR, PARAM, DOMAIN, RANGE >::workspace::task Class Reference

Inheritance diagram for parallelUtil::parallelFunctor< FUNCTOR, PARAM, DOMAIN, RANGE >::workspace::task:



Public Member Functions

- virtual bool **writeBinary** (abstractCommHandle *h) const throw ()
- virtual bool **readBinary** (abstractCommHandle *h) throw ()
- virtual void **write** (std::ostream &os) const throw ()
- void **init** (size_t ntask_, const parameters &param_, const domain &arg_)

Public Attributes

- size_t **ntask**
- parameters **param**
- domain **arg**

template<class FUNCTOR, class PARAM, class DOMAIN, class RANGE> class parallelUtil::parallelFunctor< FUNCTOR, PARAM, DOMAIN, RANGE >::workspace::task

1.7.138 python_mpi::CGC_proxy Class Reference

A class solely designed to conceal the details of providing a single CG controller for the process-pool. For this initial implementation, the _mode_ of the single controller is SLAVE_POOL, with no privision to change the mode.

Public Member Functions

- coarse_grained_controller & CGC (void)

  A reference to the static controller for the process-pool.

- const coarse_grained_controller & CGC (void) const

  A reference to the static controller for the process-pool.

- void track_module (const std::string &module_name, bool load=true) const throw (commUtil::comm_error)

  Calls the "coarse_grained_controller::track_module" method of the controller for the caller's process-pool.

- void track_call (const std::string &module_name, const std::string &function_name) const throw (commUtil::comm_error)

  Calls the "coarse_grained_controller::track_call" method of the controller for the caller's process-pool.

- void slave_process (void) const throw (python_error, commUtil::comm_-error, std::runtime_error)

  Calls the "coarse_grained_controller::slave_process" method of the controller for the caller's process-pool.

- int MPI_rank (void) const

  Rank of calling process in associated CG_controller's MPI communicator.

- int MPI_size (void) const

  Number of ranks participating in associated CG_controller's MPI communicator.

- const MPI::Comm & comm (void) const

  Associated CG_controller's MPI communicator.

- ~CGC_proxy (void)

  Handle all clean-up requirements for extended or embedded python applications using a single coarse_grained_controller.

- CGC_proxy (bool master_copy=false) throw (std::runtime_error)

  Handle all initialization requirements for extended or embedded python applications using a single coarse_grained_controller.

1.7.138.1 Detailed Description

A class solely designed to conceal the details of providing a single CG controller for the process-pool. For this initial implementation, the _mode_ of the single controller is SLAVE_POOL, with no privision to change the mode. Usage:

1. in the initialization method of a module implementing an external function:

```
*      python_mpi::CGC_proxy CGC();          // this accesses the CG-controller
*      CGC.track_module("module name",true); // command slave-processes to load
*
*      // if it is ever required (and apparently not ever for python):
*      CGC.track_module("module name",false); // command slave-processes to unl
*
```

2. at the start of an external function body:

```
*      python_mpi::CGC_proxy CGC();          // this accesses the CG-controller
*      CGC.track_call("function name");      // command slave-processes to call
*
```

1.7.139 TMatrix::aggregate_cc_functor< T > Class Template Reference

Classes

- class parameters

Public Types

- typedef numberTraits< T >::complexType **C**
- typedef numberTraits< T >::magnitudeType **R**
- typedef numberTraits< T >::integralType **Z**
- typedef python_util::simple_object_base::object_list **object_list**
- typedef python_util::simple_object_base::object_map **object_map**

Public Member Functions

- bool apply (const python_util::generic_object< C, R, Z > &arg, python_util::generic_object< C, R, Z > &val) const throw (python_util::python_error, std::runtime_error)

  wrapper for generic apply method:

- bool apply (const python_util::simple_object_base *arg, python_util::simple_object_base *val) const throw (python_util::python_error, std::runtime_error)

  *generic apply method WARNING: return-value must be preallocated as *simple_object<object_map>: this is a temporary work-around for the first-pass nature of the simple_object interface design.*

- bool **setParameters** (const parameters &param)

- const parameters & **getParameters** (void) const
- void **setStatusString** (const std::string &msg) const
- const std::string & **getStatusString** (void) const

template<class T> class TMatrix::aggregate_cc_functor< T >

1.7.140 TMatrix::aggregate_cc_functor< T >::parameters Class Reference

Public Types

- enum **return_value_kind** {

  return_value_none = 0, **C_s**, **C_s_i**, **C_e**,

  C_e_i }
- typedef python_util::options_map< C > **base_class**
- typedef base_class::object_list **object_list**

Public Member Functions

- const MPI::Comm & **comm** (void) const
- virtual void **update_cache** (bool derived=false, bool to_cache=true) throw (std::string)
- virtual void **valid_check** (void) const throw (std::string)
- virtual python_util::options_map< C > * **clone** (void) const throw (std::string)
- void **copy** (const parameters &other)
- parameters & **operator=** (const parameters &other)

- parameters & **operator=** (const python_util::options_map< R > &other)
- virtual void **write** (std::ostream &os) const
- void extract (const python_util::simple_object_base *src) throw (std::string)

  Initialize from a simple_object_base.*

- **parameters** (bool derived=false)
- **parameters** (const parameters &other)
- **parameters** (const python_util::options_map< C > &other)

Static Public Member Functions

- static return_value_kind **str_to_rval_enum** (const std::string &rval_name) throw (std::runtime_error)
- static void extract_rval_spec (const python_util::simple_object_base *rval_spec, std::string &name, R &theta_i, R &phi_i, fieldPolarization< C > &beta_i, R &theta_f, R &phi_f, fieldPolarization< C > &beta_f, size_t &N_parm) throw (python_util::python_error, std::runtime_error)

  *extract return-value parameters from *simple_object<object_list>.*

template<class T> class TMatrix::aggregate_cc_functor< T >::parameters

1.7.140.1 Member Function Documentation

1.7.140.2 template<class T > static void TMatrix::aggregate_cc_functor< T >::parameters::extract_rval_spec (const python_util::simple_object_base * *rval_spec*, std::string & *name*, R & *theta_i*, R & *phi_i*, fieldPolarization< C > & *beta_i*, R & *theta_f*, R & *phi_f*, fieldPolarization< C > & *beta_f*, size_t & *N_parm*) throw (python_util::python_error, std::runtime_error) [static]

extract return-value parameters from *simple_object<object_list>.

Parameters

[out] ***N_parm*** number of real or tuple parameters present (excepting the name itself)

1.7.141 TMatrix::aggregate_mc_functor< T > Class Template Reference

Classes

- class parameters

Public Types

- typedef numberTraits< T >::complexType **C**
- typedef numberTraits< T >::magnitudeType **R**
- typedef numberTraits< T >::integralType **Z**
- typedef python_util::simple_object_base::object_list **object_list**
- typedef python_util::simple_object_base::object_map **object_map**

Public Member Functions

- bool apply (const python_util::generic_object< C, R, Z > &arg, python_util::generic_object< C, R, Z > &val) const throw (python_util::python_error, std::runtime_error)

  wrapper for generic apply method:

- bool apply (const python_util::simple_object_base *arg, python_util::simple_object_base *val) const throw (python_util::python_error, std::runtime_error)

  *generic apply method WARNING: return-value must be preallocated as *simple_object<object_map>: this is a temporary work-around for the first-pass nature of the simple_object interface design.*

- bool **setParameters** (const parameters &param)
- const parameters & **getParameters** (void) const
- void **setStatusString** (const std::string &msg) const
- const std::string & **getStatusString** (void) const

template<class T> class TMatrix::aggregate_mc_functor< T >

1.7.142 TMatrix::aggregate_mc_functor< T >::parameters Class Reference

Public Types

- enum **return_value_kind** {

  return_value_none = 0, **C_s**, **C_s_j**, **C_s_i**,

  C_e, **C_e_j**, **C_e_i** }
- typedef python_util::options_map< C > **base_class**
- typedef base_class::object_list **object_list**

Public Member Functions

- virtual void **update_cache** (bool derived=false, bool to_cache=true) throw (std::string)
- virtual void **valid_check** (void) const throw (std::string)
- virtual python_util::options_map< C > * **clone** (void) const throw (std::string)
- void **copy** (const parameters &other)
- parameters & **operator=** (const parameters &other)
- parameters & **operator=** (const python_util::options_map< R > &other)
- virtual void **write** (std::ostream &os) const
- void extract (const python_util::simple_object_base *src) throw (std::string)

  Initialize from a simple_object_base.*

- **parameters** (bool derived=false)
- **parameters** (const parameters &other)
- **parameters** (const python_util::options_map< C > &other)

Static Public Member Functions

- static return_value_kind **str_to_rval_enum** (const std::string &rval_name) throw (std::runtime_error)
- static void extract_rval_spec (const python_util::simple_object_base *rval_-spec, std::string &name, size_t &j, bool &index_present, R &theta_-i, R &phi_i, fieldPolarization< C > &beta_i, R &theta_f, R &phi_-f, fieldPolarization< C > &beta_f, size_t &N_parm) throw (python_-util::python_error, std::runtime_error)

extract return-value parameters from ∗simple_object<object_list>.

template<class T> class TMatrix::aggregate_mc_functor< T >::parameters

1.7.142.1 Member Function Documentation

1.7.142.2 template<class T > static void TMatrix::aggregate_mc_functor< T >::parameters::extract_rval_spec (const python_util::simple_object_base ∗ *rval_spec*, std::string & *name*, size_t & *j*, bool & *index_present*, R & *theta_i*, R & *phi_i*, fieldPolarization< C > & *beta_i*, R & *theta_f*, R & *phi_f*, fieldPolarization< C > & *beta_f*, size_t & *N_parm*) throw (python_util::python_error, std::runtime_error) [static]

extract return-value parameters from ∗simple_object<object_list>.

Parameters

[out] ***N_parm*** number of real or tuple parameters present (excepting the name itself and the optional index parm)

1.7.143 TMatrix::clusterCentered_tmatrix< T > Class Template Reference

Classes

- class distributed_solve_workspace
- class parameters
- class workspace

Public Types

- typedef numberTraits< T >::magnitudeType **R**
- typedef std::vector< ntuple< R, 3 > > **ntupleList**

Public Member Functions

- const parameters & **getParameters** (void) const
- void **setParameters** (const parameters &newParam)
- const long & **l1_max** (void) const
- const long & **l2_max** (void) const
- bool **current** (void) const
- void **clear** (void)
- void **update** (void)
- size_t **N** (void) const
- void **add** (const tmatrix< T > &t_e, const ntuple< R, 3 > &position, const ntuple< R, 3 > &bodyAngle, const R &equRadius) throw ()
- const T & **mediumRefractiveIndex** (void) const
- T **mediumRefractiveIndex** (const T &n)
- const ntuple< R, 3 > & **position** (size_t n) const
- const ntuple< R, 3 > & **bodyAngle** (size_t n) const
- const R & **equivalentRadius** (size_t n) const
- R **equivalentRadius** (void) const
- const tmatrix< T > & **t_e** (size_t n) const
- const tmatrix< T > & **t_cc** (void) const
- void **apply** (const vectorFieldKet< T > &src, vectorFieldKet< T > &dest) const

- bool **writeBinary** (abstractCommHandle ∗fp) const
- bool **readBinary** (abstractCommHandle ∗fp)
- clusterCentered_tmatrix< T > & **operator=** (const clusterCentered_tmatrix< T > &other)
- **clusterCentered_tmatrix** (const clusterCentered_tmatrix &other)

Static Public Member Functions

- static clusterCentered_tmatrix< T > **pair** (const tmatrix< T > &t_e0, const tmatrix< T > &t_i0, const ntuple< R, 3 > &p0, const ntuple< R, 3 > &a0, const R &r0, const tmatrix< T > &t_e1, const tmatrix< T > &t_i1, const ntuple< R, 3 > &p1, const ntuple< R, 3 > &a1, const R &r1) throw ()
- static clusterCentered_tmatrix< T > **uniformAggregate** (const tmatrix< T > &t_e, const tmatrix< T > &t_i, const std::vector< ntuple< R, 3 > > &vP, const R &r)

Protected Member Functions

- void **set_l1_max** (long new_l1_max)
- bool **current** (bool state)
- void **calculate_l1_max** (void) throw ()
- void **calculate_l2_max** (void) throw ()
- void **adjustCentroid** (void) throw ()
- workspace & **ws** (void) const
- void **update_init** (void)
- void **sendInitShare** (abstractCommHandle ∗h) const throw ()
- void **transferWorkspaceResults** (void)

- void **sendFinishShare** (abstractCommHandle ∗h) const throw ()
- void **receiveInitShare** (abstractCommHandle ∗h) throw ()
- void **receiveFinishShare** (abstractCommHandle ∗h) throw ()

Protected Attributes

- std::vector< ntuple< R, 3 > > **vP_**
- std::vector< ntuple< R, 3 > > **vA_**
- std::vector< R > **vR_**
- T **mediumRefractiveIndex_**
- std::vector< tmatrix< T > > **vT_e_**
- tmatrix< T > **t_cc_**
- workspace ∗ **pws_**

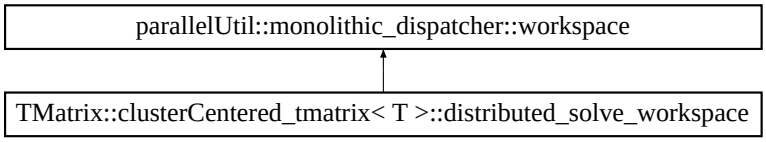
Friends

- class **distributed_solve_workspace**

template<class T> class TMatrix::clusterCentered_tmatrix< T >

1.7.144 TMatrix::clusterCentered_tmatrix< T >::distributed_solve_-workspace Class Reference

Inheritance diagram for TMatrix::clusterCentered_tmatrix< T >::distributed_-solve_workspace:



Public Types

- typedef std::vector< std::pair< size_t, size_t > > **pair_vector**
- typedef parallelUtil::monolithic_dispatcher::workspace **base_class**

Public Member Functions

- void **setOwner** (clusterCentered_tmatrix< T > &newOwner)
- clusterCentered_tmatrix< T > & **owner** (void)
- const clusterCentered_tmatrix< T > & **owner** (void) const
- void **setDispatcher** (const parallelUtil::monolithic_dispatcher &newDispatcher)
- const parallelUtil::monolithic_dispatcher & **dispatcher** (void) const
- virtual void **monolithicMPIBlock** (void) throw ()
- virtual void **finish** (void)
- virtual void **sendInitShare** (abstractCommHandle *h) const throw ()
- virtual void **sendFinishShare** (abstractCommHandle *h) const throw ()
- virtual void **receiveInitShare** (abstractCommHandle *h) throw ()
- virtual void **receiveFinishShare** (abstractCommHandle *h) throw ()
- virtual void **init** (bool isDispatcher) throw ()
- virtual void **finish** (bool isDispatcher) throw ()
- virtual base_class * **clone** (void) throw ()

- **distributed_solve_workspace** (clusterCentered_tmatrix< T > &owner, const parallelUtil::monolithic_dispatcher &dispatcher_)

template<class T> class TMatrix::clusterCentered_tmatrix< T >::distributed_solve_workspace

1.7.145 TMatrix::clusterCentered_tmatrix< T >::parameters Class Reference

Public Member Functions

- bool **writeBinary** (abstractCommHandle *h) const throw ()
- bool **readBinary** (abstractCommHandle *h) throw ()

Public Attributes

- long **convergence_l2_max**
- distributedSolverFunctor< T >::parameters::SOLVER_KIND **eSolver**
- clusterCentered_tmatrix< T >::R **solverTolerance**
- clusterCentered_tmatrix< T >::R **particleTolerance**
- clusterCentered_tmatrix< T >::R **clusterTolerance**
- size_t **maxIterations**
- size_t **blocksize**

template<class T> class TMatrix::clusterCentered_tmatrix< T >::parameters

1.7.146 TMatrix::clusterCentered_tmatrix< T >::workspace Class Reference

Public Member Functions

- void **setOwner** (clusterCentered_tmatrix< T > &newOwner)
- clusterCentered_tmatrix< T > & **owner** (void)
- const clusterCentered_tmatrix< T > & **owner** (void) const
- const long & **l2_used** (void) const
- void **set_l2_used** (long new_l2)
- void **clear** (void)
- bool **writeBinary** (abstractCommHandle *fp, bool forceResident=true) const
- bool **readBinary** (abstractCommHandle *fp)
- workspace & **operator=** (const workspace &other)
- **workspace** (const workspace &other)

Public Attributes

- gmm::dense_matrix< T > aT_cc

  Uncompressed cluster-centered tmatrix(VSW-block flattened form):

  $$\begin{bmatrix} VSW\_M : VSW\_M & VSW\_M : VSW\_N \\ VSW\_N : VSW\_M & VSW\_N : VSW\_N \end{bmatrix}$$

  where each block is subindexed using tmatrixIndex::monolithicIndex.

- std::vector< factoredPropagator< T > > **vRgU_0_i**
- std::vector< size_t > **v_i_inverse**
- std::vector< gmm::dense_matrix< T > > **vT_e**
- pthread_mutex_t **mutex**

template<class T> class TMatrix::clusterCentered_tmatrix< T >::workspace

1.7.147 TMatrix::constants::permittivityData Class Reference

Classes

- class formula
- class H_2O_formula
- class MaxwellGarnett_formula
- class mixing_formula
- class Sellmeier_formula

Public Types

- enum **material** {

  None = 0, **Cu**, **Ag**, **Au**,

  H_2O, **H_2O_Zolotarev**, **Sellmeier**, **MaxwellGarnett** }

Static Public Member Functions

- static material **materialFromString** (const std::string &name) throw ()
- static std::string **stringFromMaterial** (material e) throw ()
- static bool **usesFormula** (material eMaterial) throw ()
- template<class C , class R >

  static void **get** (material eMaterial, formula< C, R > *&pFormula, const std::vector< C > &coeff=*reinterpret_cast< std::vector< C > * >(NULL)) throw ()

- template<class C , class R >

  static void **get** (material eMaterial, std::vector< R > &vAbscissa, std::vector< C > &vOrdinate) throw ()
- template<class C , class R >

  static void **get** (material eMaterial, std::vector< R > &vAbscissa, std::vector< C > &vOrdinate, R &omega_p2, R &gamma, R &v_f) throw ()
- static double **omega_p_sqr** (const double &N, const double &m_o)

Static Public Attributes

- static const double **M_Cu**
- static const double **rho_Cu**
- static const double **M_Ag**
- static const double **rho_Ag**
- static const double **M_Au**
- static const double **rho_Au**
- static const double **Cu_v_f**
- static const double **Ag_v_f**
- static const double **Au_v_f**
- static const double **Cu_N**
- static const double **Ag_N**
- static const double **Au_N**
- static const double **Cu_m_o**
- static const double **Cu_tau**
- static const double **Ag_m_o**

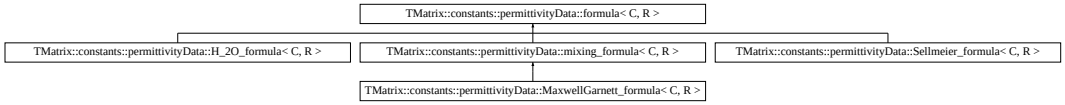
- static const double **Ag_tau**
- static const double **Au_m_o**
- static const double **Au_tau**

Static Protected Attributes

- static const size_t **E_Christy_length** = 49
- static const double **E_Christy** [E_Christy_length]
- static const double **Cu_n_Christy** [E_Christy_length][2]
- static const double **Ag_n_Christy** [E_Christy_length][2]
- static const double **Au_n_Christy** [E_Christy_length][2]
- static const size_t **H_2O_n_Zolotarev_length** = 136
- static const double **H_2O_n_Zolotarev** [H_2O_n_Zolotarev_length][3]

1.7.148 TMatrix::constants::permittivityData::formula< C, R > Class Template Reference

Inheritance diagram for TMatrix::constants::permittivityData::formula< C, R >:



Public Member Functions

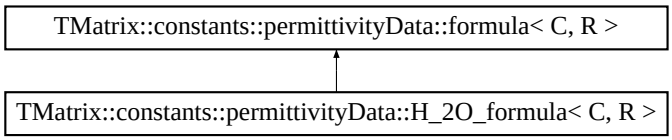
- virtual bool **inDomain** (const R &arg) const =0
- virtual bool **apply** (const R &arg, C &val, C &ferror) const =0 throw ()

- bool **apply** (const std::vector< R > &vSrc, std::vector< C > &vDest, C &ferror) const throw ()
- void **setStatusString** (const std::string &sMsg) const
- const std::string & **getStatusString** (void) const
- const std::vector< C > & **coeff** (void) const
- **formula** (const std::vector< C > &coeff)

template<class C, class R> class TMatrix::constants::permittivityData::formula< C, R >

1.7.149 TMatrix::constants::permittivityData::H_2O_formula< C, R > Class Template Reference

Inheritance diagram for TMatrix::constants::permittivityData::H_2O_formula< C, R >:



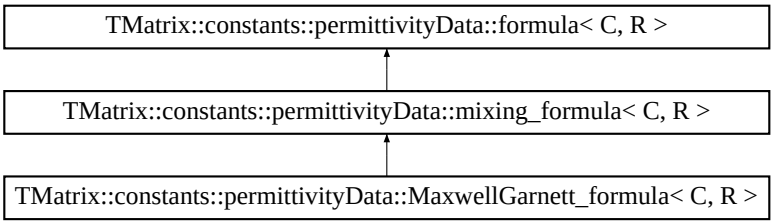
Public Member Functions

- virtual bool **inDomain** (const R &arg) const
- virtual bool **apply** (const R &arg, C &val, C &ferror) const throw ()

template<class C, class R> class TMatrix::constants::permittivityData::H_2O_formula< C, R >

1.7.150 TMatrix::constants::permittivityData::MaxwellGarnett_formula< C, R > Class Template Reference

Inheritance diagram for TMatrix::constants::permittivityData::MaxwellGarnett_formula< C, R >:



Public Types

- typedef mixing_formula< C, R > **base_class**

Public Member Functions

- bool **size_correction** (void) const
- void **set_size_correction** (bool flag)
- virtual bool **inDomain** (const R &E) const
- virtual bool **apply** (const R &E, C &val, C &ferror) const throw ()
- **MaxwellGarnett_formula** (const std::vector< C > &coeff, const std::vector< typename permittivityFunctor< C >::parameters > &vparam, bool size_correction)

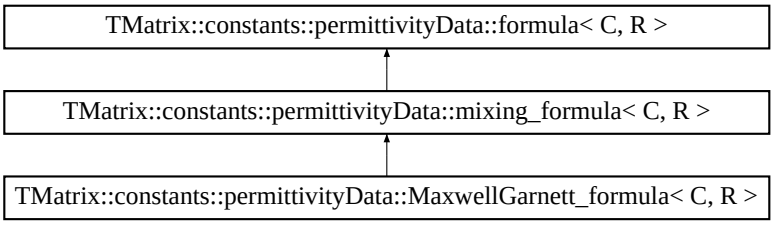
Static Public Member Functions

- static C **formula** (const C &epsilon_i, const C &epsilon_m, const R &f, const R &r, const R &lambda_, bool size_correction=false)

template<class C, class R> class TMatrix::constants::permittivityData::MaxwellGarnett_formula< C, R >

1.7.151 TMatrix::constants::permittivityData::mixing_formula< C, R > Class Template Reference

Inheritance diagram for TMatrix::constants::permittivityData::mixing_formula< C, R >:



Public Types

- typedef formula< C, R > **base_class**

Public Member Functions

- virtual bool **inDomain** (const R &arg) const =0
- virtual bool **apply** (const R &arg, C &val, C &ferror) const =0 throw ()
- **mixing_formula** (const std::vector< C > &coeff, const std::vector< typename permittivityFunctor< C >::parameters > &vparam)

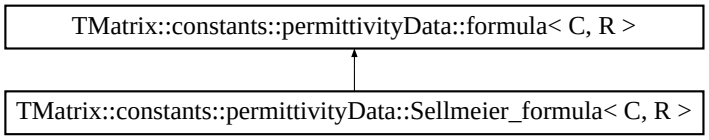
Protected Member Functions

- const permittivityFunctor< C >::parameters & **param** (size_t n) const throw ()

template<class C, class R> class TMatrix::constants::permittivityData::mixing_formula< C, R >

1.7.152 TMatrix::constants::permittivityData::Sellmeier_formula< C, R > Class Template Reference

Inheritance diagram for TMatrix::constants::permittivityData::Sellmeier_formula< C, R >:



Public Types

- typedef formula< C, R > **base_class**

Public Member Functions

- virtual bool **inDomain** (const R &arg) const
- virtual bool **apply** (const R &arg, C &val, C &ferror) const throw ()
- **Sellmeier_formula** (const std::vector< C > &coeff)

template<class C, class R> class TMatrix::constants::permittivityData::Sellmeier_-formula< C, R >

1.7.153 TMatrix::factoredPropagator< T > Class Template Reference

Public Types

- typedef numberTraits< T >::magnitudeType **R**
- typedef size_t(* **indexFoop**)(long, long, size_t)
- typedef void(* **inverseIndexFoop**)(size_t, size_t, long &, long &)
- typedef gmm::row_matrix< gmm::slvector< T > > **operatorMatrixType**

Public Member Functions

- void **applyLeft** (const blockMatrix< T > &bT, size_t l1_max, size_t l2_-max, blockMatrix< T > &bDest) const throw ()
- void **applyRight** (const blockMatrix< T > &bT, size_t l1_max, size_t l2_-max, blockMatrix< T > &bDest) const throw ()
- void **applyLeft** (const blockVector< T > &v, size_t l1_max, size_t l2_max, blockVector< T > &vDest) const throw ()
- template<class MAT1 , class MAT2 >
  void **applyRight** (const MAT1 &M, size_t l1_max, size_t l2_max, const MAT2 &dest_) const throw ()
- template<class U1 , class U2 >
  void **applyLeft** (const U1 &M, size_t l1_max, size_t l2_max, const U2 &dest_) const throw ()

- template<class MAT1 , class MAT2 >

  void **applyLeft** (const MAT1 &M, size_t l1_max, size_t l2_max, const MAT2 &dest_, gmm::abstract_matrix) const throw ()
- template<class VEC1 , class VEC2 >

  void **applyLeft** (const VEC1 &V, size_t l1_max, size_t l2_max, const VEC2 &dest_, gmm::abstract_vector) const throw ()
- template<class MAT1 , class MAT2 >

  void **applyRight** (const MAT1 &M, size_t l1_max, size_t l2_max, bool bH, const MAT2 &dest_) const throw ()
- template<class U1 , class U2 >

  void **applyLeft** (const U1 &M, size_t l1_max, size_t l2_max, bool bH, const U2 &dest_) const throw ()
- template<class MAT1 , class MAT2 >

  void **applyLeft** (const MAT1 &M, size_t l1_max, size_t l2_max, bool bH, const MAT2 &dest_, gmm::abstract_matrix) const throw ()
- template<class VEC1 , class VEC2 >

  void **applyLeft** (const VEC1 &V, size_t l1_max, size_t l2_max, bool bH, const VEC2 &dest_, gmm::abstract_vector) const throw ()
- void **clear** (void)
- void **write** (std::ostream &os) const
- bool **writeBinary** (abstractCommHandle ∗fp) const
- bool **readBinary** (abstractCommHandle ∗fp)
- size_t **binarySize** (void) const
- const std::pair< REGULARITY_KIND, REGULARITY_KIND > & **reg12** (void) const

- std::pair< REGULARITY_KIND, REGULARITY_KIND > **reg12** (const std::pair< REGULARITY_KIND, REGULARITY_KIND > &reg12New)
- long **l1_max** (void) const
- long **l1_max** (long l1_max__)
- long **l2_max** (void) const
- long **l2_max** (long l2_max__)
- operatorMatrixType & **rotator** (void)
- const operatorMatrixType & **rotator** (void) const
- operatorMatrixType & **translatorH** (void)
- const operatorMatrixType & **translatorH** (void) const
- operatorMatrixType & **translatorK** (void)
- const operatorMatrixType & **translatorK** (void) const

Static Public Member Functions

- template<class MAT1 , class MAT2 , class MAT3 >
  static void **multRightDiagL** (const MAT1 &srcLeft, const MAT2 &srcRight, size_t l1_max, size_t l2_max, const MAT3 &dest_) throw ()
- template<class MAT1 , class U2 , class U3 >
  static void **multLeftDiagL** (const MAT1 &srcLeft, const U2 &srcRight, size_t l1_max, size_t l2_max, const U3 &dest_) throw ()
- template<class MAT1 , class MAT2 , class MAT3 >
  static void **multLeftDiagL** (const MAT1 &srcLeft, const MAT2 &srcRight, size_t l1_max, size_t l2_max, const MAT3 &dest_, gmm::abstract_matrix) throw ()

- template<class MAT1 , class MAT2 , class MAT3 >

  static void **multLeftDiagL** (const MAT1 &srcLeft, const MAT2 &srcRight, size_t l1_max, size_t l2_max, const MAT3 &dest_, gmm::row_major, gmm::row_major) throw ()

- template<class MAT1 , class MAT2 , class MAT3 , class PO2 , class PO3 >

  static void **multLeftDiagL** (const MAT1 &srcLeft, const MAT2 &srcRight, size_t l1_max, size_t l2_max, const MAT3 &dest_, PO2, PO3) throw ()

- template<class MAT1 , class VEC2 , class VEC3 >

  static void **multLeftDiagL** (const MAT1 &srcLeft, const VEC2 &srcRight, size_t l1_max, size_t l2_max, const VEC3 &dest_, gmm::abstract_vector) throw ()

- template<class MAT1 , class MAT2 , class MAT3 >

  static void **multRightDiagM** (const MAT1 &srcLeft, const MAT2 &srcRight, size_t l1_max, size_t l2_max, const MAT3 &dest_) throw ()

- template<class MAT1 , class U2 , class U3 >

  static void **multLeftDiagM** (const MAT1 &srcLeft, const U2 &srcRight, size_t l1_max, size_t l2_max, const U3 &dest_) throw ()

- template<class MAT1 , class MAT2 , class MAT3 >

  static void **multLeftDiagM** (const MAT1 &srcLeft, const MAT2 &srcRight, size_t l1_max, size_t l2_max, const MAT3 &dest_, gmm::abstract_matrix) throw ()

- template<class MAT1 , class VEC2 , class VEC3 >

  static void **multLeftDiagM** (const MAT1 &srcLeft, const VEC2 &srcRight, size_t l1_max, size_t l2_max, const VEC3 &dest_, gmm::abstract_vector) throw ()

- static void **convertFactoredToBlockForm** (const factoredPropagator< T > &src, std::vector< long > &vnLayerRows, std::vector< long > &vnLayerCols, blockMatrix< T > &bRot, blockMatrix< T > &bTrans, blockMatrix< T > &bInvRot) throw ()
- static void init (const T &x, const R &u, const R &p, long l1_max, long l2_max, const std::pair< REGULARITY_KIND, REGULARITY_KIND > &reg12, factoredPropagator< T > &dest) throw ()

  this version assumes modified spherical co-ordinates: $r \rightarrow x, \cos(\theta) \rightarrow u, \phi \rightarrow p$

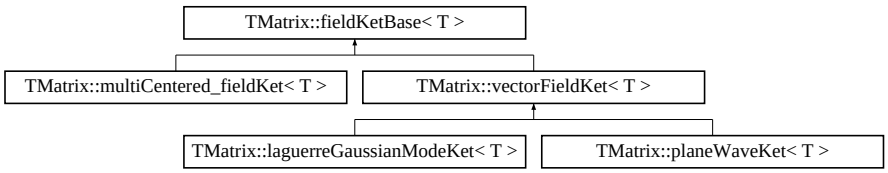
- static void init (const ntuple< R, 3 > &p, const T &mediumRefractiveIndex, long l1_max, long l2_max, const std::pair< REGULARITY_KIND, REGULARITY_KIND > &reg12, factoredPropagator< T > &dest) throw ()

  this version assumes "p" in spherical co-ordinates: $p_0 \rightarrow r, p_1 \rightarrow \theta, p_2 \rightarrow \phi$

template<class T> class TMatrix::factoredPropagator< T >

1.7.154 TMatrix::fieldKetBase< T > Class Template Reference

Inheritance diagram for TMatrix::fieldKetBase< T >:



Public Types

- enum **field_ket_kind** { **ABSTRACT_KET**, **SINGLE_CENTER_KET**, **MULTI_CENTER_KET** }

Public Member Functions

- virtual field_ket_kind **kind** (void) const =0
- virtual bool **readBinary** (abstractCommHandle ∗fp)=0
- virtual bool **writeBinary** (abstractCommHandle ∗fp) const =0
- virtual fieldKetBase ∗ **clone** (void) const =0
- virtual bool **is_multi_centered** (void) const

Static Public Member Functions

- static bool **readBinaryVirtual** (abstractCommHandle ∗fp, fieldKetBase< T > ∗&pket)
- static bool **writeBinaryVirtual** (abstractCommHandle ∗fp, const fieldKetBase< T > ∗pket)

template<class T> class TMatrix::fieldKetBase< T >

1.7.155 TMatrix::fieldPolarization< T > Class Template Reference

Public Types

- enum **polarization_kind** {

  none = 0, **minus_helical** = 1, **plus_helical**, **longitudinal_helical**,

horizontal, **vertical**, **N_polarization_kind** = 5 }

Public Member Functions

- const T & **e_m** (void) const
- const T & **e_0** (void) const
- const T & **e_p** (void) const
- const T **x** (void) const
- const T **y** (void) const
- const T **z** (void) const
- const std::vector< T > **vect** (void)
- void **sphereForm** (const T &theta_k, const T &phi_k, T &beta_r, T &beta_t, T &beta_p) const throw ()
- void **sphereForm** (const T &theta_k, const T &phi_k, std::vector< T > &vBeta) const throw ()
- void **cartesianForm** (std::vector< T > &vBeta) const throw ()
- void **cartesianForm** (const T &k_x, const T &k_y, const T &k_z, T &beta_x, T &beta_y, T &beta_z) const throw ()
- void **cartesianForm** (const std::vector< T > &vK, std::vector< T > &vBeta) const throw ()
- void **cartesianForm** (const T &theta_k, const T &phi_k, std::vector< T > &vBeta) const throw ()
- void **write** (std::ostream &os) const
- **fieldPolarization** (polarization_kind eKind)
- **fieldPolarization** (const T &e_m=zero< T >(), const T &e_p=one< T >(), const T &e_0=zero< T >())

- **fieldPolarization** (const fieldPolarization< T > &other)

Static Public Member Functions

- static polarization_kind **stringToPolarizationEnum** (const std::string &name) throw ()
- static std::string **polarizationEnumToString** (const polarization_kind eKind) throw ()

Protected Member Functions

- void **normalize** (void)

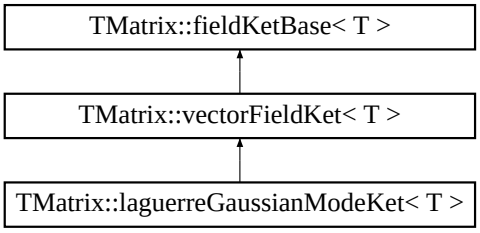
Protected Attributes

- std::vector< T > **v_**

template<class T> class TMatrix::fieldPolarization< T >

1.7.156 TMatrix::laguerreGaussianModeKet< T > Class Template Reference

Inheritance diagram for TMatrix::laguerreGaussianModeKet< T >:



Public Member Functions

- **laguerreGaussianModeKet** (const T &w0, const T &z_w, const T &U0=one< T >(), const fieldPolarization< T > &beta=fieldPolarization< T >(), const long &lMax=100) throw ()
- **laguerreGaussianModeKet** (const T &w0, const T &z_w, const long &l, const long &m, const T &U0=one< T >(), const fieldPolarization< T > &beta=fieldPolarization< T >(), const long &lMax=100) throw ()

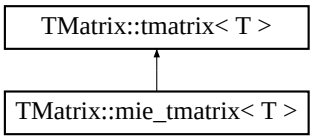
Friends

- void **tmatrix** (const vectorFieldKet< T > &src, vectorFieldKet< T > &dest) const

template<typename T> class TMatrix::laguerreGaussianModeKet< T >

1.7.157 TMatrix::mie_tmatrix< T > Class Template Reference

Inheritance diagram for TMatrix::mie_tmatrix< T >:



Public Types

- typedef numberTraits< T >::magnitudeType **R**

Public Member Functions

- **mie_tmatrix** (const T &x_s, const T &x, const long &lMax=100, bool internalFields=false) throw ()
- **mie_tmatrix** (const T &x_s, const T &x, const vectorFieldKet< T > &k, bool internalFields=false) throw ()

Static Public Member Functions

- static void **init** (const T &x_s, const T &x, const long &lMax, tmatrix< T > &t, tmatrix< T > &t_int)
- static void **init** (const T &x_s, const T &x, const vectorFieldKet< T > &k, tmatrix< T > &t, tmatrix< T > &t_int)
- static void **init** (const T &x_s_, const T &x_, bool internalFields_, long l_max, std::vector< T > &dest) throw ()

Protected Attributes

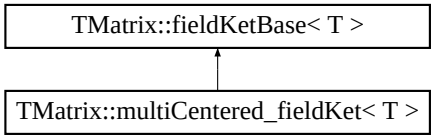
- T **x_s_**
- T **x_**

- bool **internalFields_**

template<typename T> class TMatrix::mie_tmatrix< T >

1.7.158 TMatrix::multiCentered_fieldKet< T > Class Template Reference

Inheritance diagram for TMatrix::multiCentered_fieldKet< T >:



Public Types

- typedef numberTraits< T >::magnitudeType **R**

Public Member Functions

- virtual fieldKetBase< T >::field_ket_kind **kind** (void) const
- virtual bool **readBinary** (abstractCommHandle *fp)
- virtual bool **writeBinary** (abstractCommHandle *fp) const
- virtual fieldKetBase< T > * **clone** (void) const
- virtual bool **is_multi_centered** (void) const
- const std::vector< vectorFieldKet< T > > & **kets** (void) const
- std::vector< vectorFieldKet< T > > & **kets** (void)
- const std::vector< ntuple< R, 3 > > & **positions** (void) const

- std::vector< ntuple< R, 3 > > & **positions** (void)
- size_t **size** (void) const
- void **resize** (size_t n)
- void **clear** (void)
- bool **compatible** (const multiCentered_fieldKet< T > &other) const
- bool **compatible** (const std::vector< ntuple< R, 3 > > &otherPos) const
- multiCentered_fieldKet & **operator+=** (const multiCentered_fieldKet &other)
- multiCentered_fieldKet & **operator-=** (const multiCentered_fieldKet &other)
- multiCentered_fieldKet & **operator*=** (const T &factor)
- **multiCentered_fieldKet** (size_t size_)
- **multiCentered_fieldKet** (const multiCentered_fieldKet< T > &other)
- **multiCentered_fieldKet** (const std::vector< vectorFieldKet< T > > &kets, const std::vector< ntuple< R, 3 > > &positions)

Static Public Member Functions

- static multiCentered_fieldKet< T > **planeWaveKet** (const std::vector< ntuple< R, 3 > > &positions, const fieldPolarization< T > &beta, long l_max)

template<class T> class TMatrix::multiCentered_fieldKet< T >

1.7.159 TMatrix::multiCentered_tmatrix< T > Class Template Reference

Classes

- class workspace

Public Types

- typedef numberTraits< T >::magnitudeType **R**
- typedef std::vector< ntuple< R, 3 > > **ntupleList**

Public Member Functions

- const long & **l_max** (void) const
- bool **current** (void) const
- void **clear** (void)
- void **update** (void) const
- size_t **N** (void) const
- void **add** (const tmatrix< T > &t_e, const tmatrix< T > &t_i, const ntuple< R, 3 > &position, const ntuple< R, 3 > &bodyAngle, const R &equRadius, const regionSelector< T > &region) throw ()
- const T & **mediumRefractiveIndex** (void) const
- T **mediumRefractiveIndex** (const T &n)
- const ntuple< R, 3 > & **position** (size_t n) const
- const std::vector< ntuple< R, 3 > > & **positions** (void) const
- const ntuple< R, 3 > & **bodyAngle** (size_t n) const
- const std::vector< ntuple< R, 3 > > & **bodyAngles** (void) const
- const R & **equivalentRadius** (size_t n) const

- const std::vector< R > & **equivalentRadii** (void) const
- const regionSelector< T > & **region** (size_t n) const
- R **equivalentRadius** (void) const
- const tmatrix< T > & **t_e** (size_t n) const
- const tmatrix< T > & **t_i** (size_t n) const
- tmatrix< T > **t_N** (size_t i, size_t j) const
- void **apply** (const vectorFieldKet< T > &src, vectorFieldKet< T > &dest) const throw ()
- void **apply** (const multiCentered_fieldKet< T > &src, multiCentered_-fieldKet< T > &dest_ext, std::vector< multiCentered_fieldKet< T > > &dest_int) const throw ()
- R **Q_s** (const T &theta_i, const T &phi_i, const fieldPolarization< T > &beta_i, const T &theta_f, const T &phi_f, const fieldPolarization< T > &beta_f, std::vector< R > *partials=NULL) const throw ()
- R **Q_s** (const T &theta_i, const T &phi_i, const fieldPolarization< T > &beta_i, const fieldPolarization< T > &beta_f, std::vector< R > *partials=NULL) const throw ()
- R **Q_s** (const T &theta_i, const T &phi_i, const fieldPolarization< T > &beta_i, std::vector< R > *partials=NULL) const throw ()
- R **Q_e** (const T &theta_i, const T &phi_i, const fieldPolarization< T > &beta_i, std::vector< R > *partials=NULL) const throw ()
- R **Q_s** (const T &theta_i, const T &phi_i, std::vector< R > *partials=NULL) const throw ()
- R **Q_e** (const T &theta_i, const T &phi_i, std::vector< R > *partials=NULL) const throw ()

- R **Q_s** (std::vector< R > *partials=NULL) const throw ()
- R **Q_e** (std::vector< R > *partials=NULL) const throw ()
- R **Q_s_j** (size_t j, const T &t_i, const T &p_i, const fieldPolarization< T > &beta_i, const T &t_f, const T &p_f, const fieldPolarization< T > &beta_f) const throw ()
- R **Q_s_j** (size_t j, const T &t_i, const T &p_i, const fieldPolarization< T > &beta_i) const throw ()
- R **Q_e_j** (size_t j) const throw ()
- bool **writeBinary** (abstractCommHandle *fp) const
- bool **readBinary** (abstractCommHandle *fp)
- multiCentered_tmatrix< T > & **operator=** (const multiCentered_tmatrix< T > &other)
- **multiCentered_tmatrix** (const multiCentered_tmatrix &other)

Static Public Member Functions

- static void **write_fields_data** (const std::string &fileName, const multiCentered_tmatrix< T > &t_mc, const T &n_m, const R &lambda_0) throw ()
- static void **read_fields_data** (const std::string &fileName, multiCentered_-tmatrix< T > &t_mc, T &n_m, R &lambda_0) throw ()
- static multiCentered_tmatrix< T > **pair** (const tmatrix< T > &t_e0, const tmatrix< T > &t_i0, const ntuple< R, 3 > &p0, const ntuple< R, 3 > &a0, const R &r0, const regionSelector< T > &region0, const tmatrix< T > &t_-e1, const tmatrix< T > &t_i1, const ntuple< R, 3 > &p1, const ntuple< R, 3 > &a1, const R &r1, const regionSelector< T > &region1) throw ()

- static multiCentered_tmatrix< T > **uniformAggregate** (const tmatrix< T > &t_e, const tmatrix< T > &t_i, const std::vector< ntuple< R, 3 > > &vP, const R &r, const regionSelector< T > &region)

Protected Member Functions

- void **free_regionp** (void)
- long **l_max** (long new_l_max)
- bool **current** (bool state) const
- workspace & **ws** (void) const

Protected Attributes

- std::vector< ntuple< R, 3 > > **vP_**
- std::vector< ntuple< R, 3 > > **vA_**
- std::vector< R > **vR_**
- std::vector< regionSelector< T > * > **vRegionp_**
- T **mediumRefractiveIndex_**
- std::vector< tmatrix< T > > **vT_e_**
- std::vector< tmatrix< T > > **vT_i_**
- workspace * **pws_**

template<class T> class TMatrix::multiCentered_tmatrix< T >

1.7.160 TMatrix::multiCentered_tmatrix< T >::workspace Class Reference

Classes

- class zero_constructor

Public Member Functions

- void **setOwner** (multiCentered_tmatrix< T > &newOwner)
- multiCentered_tmatrix< T > & **owner** (void)
- void **identity** (blockMatrix< T > &bI, const std::pair< REGULARITY_KIND, REGULARITY_KIND > &reg12) const throw ()
- void **zero** (blockMatrix< T > &bZ, const std::pair< REGULARITY_KIND, REGULARITY_KIND > &reg12) const throw ()
- void **alloc_zero_blocks** (blockMatrix< T > &u) const throw ()
- void **clear** (void)
- bool **writeBinary** (abstractCommHandle *fp) const
- bool **readBinary** (abstractCommHandle *fp)

Public Attributes

- std::vector< long > **vnLayerRows**
- std::vector< long > **vnLayerCols**
- compressed2DIndex **compressor**
- gmm::abstract_block_resident_matrix< blockMatrix< T >, 0 > **aT**
- std::vector< factoredPropagator< T > > **vU_i_n**
- std::vector< factoredPropagator< T > > **vU_n_i**
- std::vector< blockMatrix< T > > **vT_e**

- std::vector< blockMatrix< T > > **vT_i**
- size_t **currentParticle**
- REGULARITY_KIND **eReg1**
- REGULARITY_KIND **eReg2**
- T **theta_i**
- T **phi_i**
- T **theta_f**
- T **phi_f**
- fieldPolarization< T > **beta_i**
- fieldPolarization< T > **beta_f**
- std::vector< blockVector< T > > **vvf_i**
- std::vector< blockVector< T > > **vvp_i**
- std::vector< long > **vnTrRows**
- std::vector< long > **vnTrCols**
- T **scalarSum**
- blockMatrix< T > **bSum**
- std::vector< R > * **partials**
- gmm::abstract_block_resident_matrix< blockMatrix< T >, 0 > **aTU**
- blockVector< T > **k_ip**
- const multiCentered_fieldKet< T > * **psrc_ket**
- multiCentered_fieldKet< T > * **pdest_ket_ext**
- multiCentered_fieldKet< T > * **pdest_ket_int**
- pthread_mutex_t **mutex**
- parallelUtil::multiMutex **aT_mutex**

template<class T> class TMatrix::multiCentered_tmatrix< T >::workspace

1.7.161 TMatrix::multiCentered_tmatrix< T >::workspace::zero_constructor Class Reference

Public Types

- typedef type_util::universal_constructor< blockMatrix< T > > **base_class**

Public Member Functions

- virtual blockMatrix< T > **construct** (void) const
- virtual blockMatrix< T > * **heap_construct** (void) const
- virtual void **resize** (blockMatrix< T > &u) const
- virtual bool **operator==** (const base_class &uc) const
- bool **operator!=** (const base_class &uc) const
- virtual base_class * **clone** (void) const
- **zero_constructor** (const zero_constructor &other)
- **zero_constructor** (const multiCentered_tmatrix< T >::workspace &owner, const std::pair< REGULARITY_KIND, REGULARITY_KIND > &reg12)

template<class T> class TMatrix::multiCentered_tmatrix< T >::workspace::zero_constructor

1.7.162 TMatrix::multiCentered_vectorFieldKet< T > Struct Template Reference

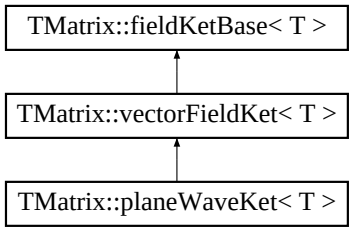
Public Member Functions

- multiCentered_vectorFieldKet & **operator+=** (const multiCentered_vectorFieldKet &other)
- multiCentered_vectorFieldKet & **operator-=** (const multiCentered_vectorFieldKet &other)
- multiCentered_vectorFieldKet & **operator*=** (const T &factor)
- void **copy** (const multiCentered_vectorFieldKet &other)
- multiCentered_vectorFieldKet & **operator=** (const multiCentered_vectorFieldKet &other)
- **multiCentered_vectorFieldKet** (const multiCentered_vectorFieldKet &other)

template<class T> struct TMatrix::multiCentered_vectorFieldKet< T >

1.7.163 TMatrix::planeWaveKet< T > Class Template Reference

Inheritance diagram for TMatrix::planeWaveKet< T >:



Public Member Functions

- **planeWaveKet** (const fieldPolarization< T > &beta=fieldPolarization< T >(), long l_max=100)

Static Public Member Functions

- static void **init** (const fieldPolarization< T > &beta, long l_max, vectorFieldKet< T > &dest)

Static Protected Member Functions

- static T **lFactor** (const long &l)

Protected Attributes

- fieldPolarization< T > **beta_**

Friends

- void **tmatrix** (const vectorFieldKet< T > &src, vectorFieldKet< T > &dest) const

template<typename T> class TMatrix::planeWaveKet< T >

1.7.164 TMatrix::position_ket< T > Struct Template Reference

Public Types

- typedef numberTraits< T >::magnitudeType **R**

Public Member Functions

- void **copy** (const position_ket &other)
- position_ket & **operator=** (const position_ket &other)
- **position_ket** (const position_ket &other)
- **position_ket** (const ntuple< R, 3 > &point_, const vectorFieldKet< T > &ket_)

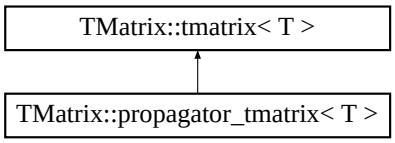
Public Attributes

- ntuple< R, 3 > **point**
- vectorFieldKet< T > **ket**

template<class T> struct TMatrix::position_ket< T >

1.7.165 TMatrix::propagator_tmatrix< T > Class Template Reference

Inheritance diagram for TMatrix::propagator_tmatrix< T >:



Classes

- class **workspace**

Public Types

- typedef numberTraits< T >::magnitudeType **R**
- typedef size_t(* **indexFoop**)(long, long, size_t)
- typedef void(* **inverseIndexFoop**)(size_t, size_t, long &, long &)

Public Member Functions

- void **invertSelf** (void) throw ()
- **propagator_tmatrix** (const T &x, const R &u, const R &p, const long &lMax=100, bool internalFields=false) throw ()
- **propagator_tmatrix** (const T &x, const R &u, const R &p, const vectorFieldKet< T > &sampleKet, bool internalFields=false) throw ()
- **propagator_tmatrix** (const T &z, const vectorFieldKet< T > &sampleKet, bool internalFields=false) throw ()

Static Public Member Functions

- static void **init** (const T &x, const R &u, const R &p, long l_max, const std::vector< std::pair< REGULARITY_KIND, REGULARITY_KIND > > &vReg2, const std::vector< long > &vM, std::vector< long > &vnLayerRows, std::vector< long > &vnLayerCols, blockMatrix< T > &bDest) throw ()
- static void **init** (const ntuple< R, 3 > &p, const T &mediumRefractiveIndex, long l_max, const std::vector< std::pair< REGULARITY_KIND, REGULARITY_KIND > > &vReg2, const std::vector< long > &vM, std::vector< long > &vnLayerRows, std::vector< long > &vnLayerCols, blockMatrix< T > &bDest) throw ()

- static void **init** (const T &x, const R &u, const R &p, long l1_max, long l2_max, const std::vector< std::pair< REGULARITY_KIND, REGULARITY_KIND > > &vReg2, const std::vector< long > &vM, std::vector< long > &vnLayerRows, std::vector< long > &vnLayerCols, blockMatrix< T > &bRot, blockMatrix< T > &bTrans, blockMatrix< T > &bInvRot) throw ()
- static void **init** (const ntuple< R, 3 > &p, const T &mediumRefractiveIndex, long l1_max, long l2_max, const std::vector< std::pair< REGULARITY_KIND, REGULARITY_KIND > > &vReg2, const std::vector< long > &vM, std::vector< long > &vnLayerRows, std::vector< long > &vnLayerCols, blockMatrix< T > &bRot, blockMatrix< T > &bTrans, blockMatrix< T > &bInvRot) throw ()
- static void **applyFactoredLeft** (const blockMatrix< T > &bRot, const blockMatrix< T > &bTrans, const blockMatrix< T > &bInvRot, const blockMatrix< T > &bT, const compressed2DIndex &compressor, blockMatrix< T > &bDest) throw ()
- static void **applyFactoredRight** (const blockMatrix< T > &bT, const blockMatrix< T > &bRot, const blockMatrix< T > &bTrans, const blockMatrix< T > &bInvRot, const compressed2DIndex &compressor, blockMatrix< T > &bDest) throw ()
- static void **applyFactoredLeft** (const blockMatrix< T > &bRot, const blockMatrix< T > &bTrans, const blockMatrix< T > &bInvRot, const blockVector< T > &v, const compressed2DIndex &compressor, blockVector< T > &vDest) throw ()
- static void **additionCoefficients** (const T &x, const R &u, const R &p, long l1_max, long l2_max, const std::vector< long > &vM, REGULARITY_-

KIND eReg1, REGULARITY_KIND eReg2, const compressed2DIndex &indexer, gmm::dense_matrix< T > &aH, gmm::dense_matrix< T > &aK) throw ()

- template<class MAT1 , class MAT2 >
  static void **additionCoefficients** (const T &x, long l1_max, long l2_max, REGULARITY_KIND eReg1, REGULARITY_KIND eReg2, indexFoop indexer, const MAT1 &aH_, const MAT2 &aK_) throw ()

Static Public Attributes

- static const long **l_min** = MIN_MULTIPOLE_ORDER

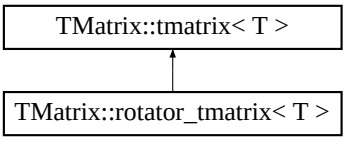
Protected Attributes

- T **x_**
- R **u_**
- R **p_**
- bool **internalFields_**

template<class T> class TMatrix::propagator_tmatrix< T >

1.7.166 TMatrix::rotator_tmatrix< T > Class Template Reference

Inheritance diagram for TMatrix::rotator_tmatrix< T >:



Public Types

- typedef numberTraits< T >::magnitudeType **R**
- typedef size_t(* **indexFoop**)(long, long, size_t)
- typedef void(* **inverseIndexFoop**)(size_t, size_t, long &, long &)

Public Member Functions

- const R & **alpha** (void)
- const R & **beta** (void)
- const R & **gamma** (void)
- void **invertSelf** (void) throw ()
- **rotator_tmatrix** (const R &alpha, const R &beta, const R &gamma, long l_max) throw ()

Static Public Member Functions

- static void **init** (const R &alpha, const R &beta, const R &gamma, long l_max, const std::vector< REGULARITY_KIND > &vReg, std::vector< long > &vnLayerRows, std::vector< long > &vnLayerCols, blockMatrix< T > &aDest) throw ()
- template<class MAT >
  static void **init** (const R &alpha, const R &beta, const R &gamma, long l_-

max, indexFoop indexer, const MAT &dest_) throw ()

- static void **init** (const ntuple< R, 3 > &a, long l_max, const std::vector< REGULARITY_KIND > &vReg, std::vector< long > &vnLayerRows, std::vector< long > &vnLayerCols, blockMatrix< T > &aDest) throw ()

Protected Attributes

- R **alpha_**
- R **beta_**
- R **gamma_**

template<typename T> class TMatrix::rotator_tmatrix< T >

1.7.167 TMatrix::scalarFieldKet< T > Class Template Reference

Public Types

- typedef indefiniteVector< T > **ket_data_type**

Public Member Functions

- gmm::linalg_traits< ket_data_type >::const_reference **operator()** (const long &l, const long &m) const
- gmm::linalg_traits< ket_data_type >::reference **operator()** (const long &l, const long &m)
- void **write** (std::ostream &os) const
- **scalarFieldKet** (const long &lMax)

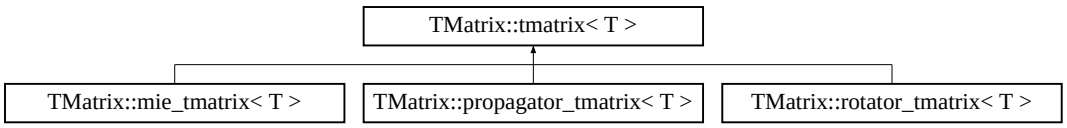
Protected Attributes

- indefiniteVector< T > **ketData_**

template<class T> class TMatrix::scalarFieldKet< T >

1.7.168 TMatrix::tmatrix< T > Class Template Reference

Inheritance diagram for TMatrix::tmatrix< T >:



Classes

- class **workspace**

Public Types

- typedef size_t **size_type**
- typedef indefiniteSparseMatrix< T > **fieldClassLayerMatrixType**
- typedef numberTraits< T >::magnitudeType **R**

Public Member Functions

- void **getCommonIndexVectors** (std::vector< long > &vnLayerRows, std::vector< long > &vnLayerCols) const throw ()

- long **lMax** (void) const
- void **nnzByIndex** (std::map< long, long > &mL, std::map< long, long > &mM) const
- void **nnzByIndex** (std::map< nDimensionalKey< long, 2 >, long > &mM, std::map< nDimensionalKey< long, 2 >, long > &mL) const
- void **toCompressedFormat** (blockMatrix< T > &aBlocks, std::vector< long > &vnLayerRows, std::vector< long > &vnLayerCols) const throw ()
- void **fromCompressedFormat** (const blockMatrix< T > &aBlocks, const std::vector< long > &vnLayerRows, const std::vector< long > &vnLayer-Cols) throw ()
- void **toDenseFormat** (gmm::dense_matrix< T > &dest) const throw ()
- void **fromDenseFormat** (const gmm::dense_matrix< T > &src) throw ()
- const fieldClassLayerMatrixType & **operator()** (const VSW_KIND &eVSW1, const REGULARITY_KIND &eReg1, const VSW_KIND &eVSW2, const REGULARITY_KIND &eReg2) const
- fieldClassLayerMatrixType & **operator()** (const VSW_KIND &eVSW1, const REGULARITY_KIND &eReg1, const VSW_KIND &eVSW2, const REGULARITY_KIND &eReg2)
- gmm::linalg_traits< fieldClassLayerMatrixType >::const_reference **operator()** (const VSW_KIND &eVSW1, const REGULARITY_KIND &eReg1, const long &l1, const long &m1, const VSW_KIND &eVSW2, const REGULARITY_KIND &eReg2, const long &l2, const long &m2) const
- gmm::linalg_traits< fieldClassLayerMatrixType >::reference **operator()** (const VSW_KIND &eVSW1, const REGULARITY_KIND &eReg1, const long &l1, const long &m1, const VSW_KIND &eVSW2, const REGULARITY_KIND &eReg2, const long &l2, const long &m2)

- tmatrix< T > & **operator=** (const tmatrix &other)
- tmatrix & **operator+=** (const tmatrix &other)
- tmatrix & **operator-=** (const tmatrix &other)
- tmatrix & **operator∗=** (const tmatrix &other)
- tmatrix & **operator∗=** (const T &scale)
- void **apply** (const vectorFieldKet< T > &src, vectorFieldKet< T > &dest) const
- void **resize** (const long &lMax)
- void **clear** (void)
- void **clean** (const T &eps)
- void **write** (std::ostream &os, bool writeIndices=false) const
- bool **writeBinary** (abstractCommHandle ∗fp) const
- bool **readBinary** (abstractCommHandle ∗fp)
- void **writeSignature** (std::ostream &os) const
- T **Q_s** (const T &t_i, const T &p_i, const fieldPolarization< T > &beta_i, const T &t_f, const T &p_f, const fieldPolarization< T > &beta_f) const throw ()
- T **Q_s** (const T &t_i, const T &p_i, const fieldPolarization< T > &beta_i, const fieldPolarization< T > &beta_f) const throw ()
- T **Q_s** (const T &t_i, const T &p_i, const fieldPolarization< T > &beta_i) const throw ()
- T **Q_e** (const T &t_i, const T &p_i, const fieldPolarization< T > &beta_i) const throw ()
- T **Q_s** (const T &t_i, const T &p_i) const throw ()
- T **Q_e** (const T &t_i, const T &p_i) const throw ()

- T **Q_s** (void) const throw ()
- T **Q_e** (void) const throw ()
- **tmatrix** (const tmatrix &other)
- **tmatrix** (const long &lMax)
- **tmatrix** (const regionSelector< T > &region, const long &lMax, bool internalFields=false) throw ()
- **tmatrix** (const regionSelector< T > &region, const vectorFieldKet< T > &k, bool internalFields=false) throw ()
- void **regularityTransition** (REGULARITY_KIND srcReg, REGULARITY_KIND destReg, const T &x) throw ()
- void **verifyPrecision** (std::ostream &os) const throw ()
- void **verifyUnitarity** (std::ostream &os, const T &threshold=epsilon< T >(), size_type test_lMax=0, REGULARITY_KIND scatteredReg=OUTGOING) const throw ()
- void **verifySymmetry** (std::ostream &os, const T &threshold=epsilon< T >(), size_type test_lMax=0, REGULARITY_KIND scatteredReg=OUTGOING) const throw ()

Static Public Member Functions

- static void **getCommonIndexVectors** (const tmatrix< T > &t1, const tmatrix< T > &t2, std::vector< long > &vnLayerRows, std::vector< long > &vnLayerCols) throw ()
- static void **getDenseIndexVectors** (long l1_max, long l2_max, std::vector< long > &vnLayerRows, std::vector< long > &vnLayerCols) throw ()
- static void **mapsToIndexVectors** (const std::map< long, long > &mL1, const std::map< long, long > &mM1, const std::map< long, long > &mL2,

const std::map< long, long > &mM2, std::vector< long > &vnLayerRows, std::vector< long > &vnLayerCols) throw ()

- static void **mapsToIndexVector** (const std::map< long, long > &mL, const std::map< long, long > &mM, std::vector< long > &vnElts) throw ()
- static T **propagator** (const VSW_KIND &eVSW1, const REGULARITY_-KIND &eReg1, const long &l1, const long &m1, const VSW_KIND &eVSW2, const REGULARITY_KIND &eReg2, const long &l2, const long &m2, const T &x, const T &u, const T &p, long lMax=100, long specialInit=0) throw ()
- static R **physicalQ** (const R &Q, const R &r, const R &lambda)
- template<class V1 , class V2 >
  static void physicalQ (V1 &dest, const R &r, const R &lambda, const V2 &src)
  convert vector of different efficiencies at single wavelength:
- static R **physicalCrossSection** (const R &Q, const R &lambda)
- template<class V1 , class V2 >
  static void physicalCrossSection (V1 &dest, const R &lambda, const V2 &src)
  convert vector of different cross sections at single wavelength:
- static void **transpose** (const tmatrix< T > &tSrc, tmatrix< T > &tDest) throw ()
- static void **hermitianConjugate** (const tmatrix< T > &tSrc, tmatrix< T > &tDest) throw ()
- static void **init** (const regionSelector< T > &region, const std::vector< long > &vL, const std::vector< long > &vM, tmatrix< T > &tExternal, tmatrix<

T > &tInternal, tmatrix< T > *pQ=NULL, tmatrix< T > *pRgQ=NULL) throw ()

- static void **init_l_m_domainFromKet** (const vectorFieldKet< T > &k, std::vector< long > &vL, std::vector< long > &vM)

Protected Member Functions

- long **nrows** (void) const
- long **ncols** (void) const
- void **nnzByRawIndex** (std::map< long, long > &mRows, std::map< long, long > &mCols) const
- **tmatrix** (const long &nrows_, const long &ncols_)

Static Protected Member Functions

- static void **initMultipoleSpace** (const std::vector< long > &vL, const std::vector< long > &vM) throw ()

Protected Attributes

- std::vector< fieldClassLayerMatrixType > **matrixData_**

template<class T> class TMatrix::tmatrix< T >

1.7.169 TMatrix::tmatrixIndex Class Reference

Public Types

- typedef size_t **size_type**

- typedef size_t(∗ **indexFoop**)(long, long, size_t)
- typedef void(∗ **inverseIndexFoop**)(size_t, size_t, long &, long &)

Static Public Member Functions

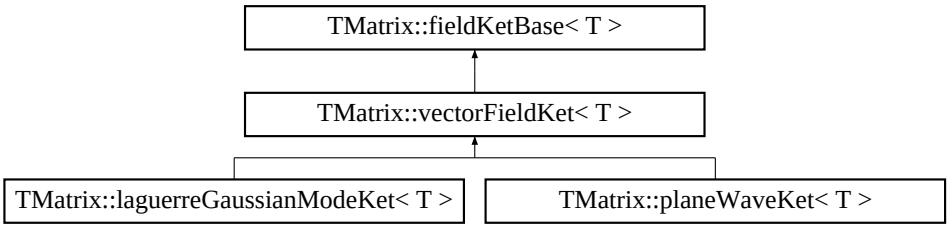
- static std::string **fieldClassDescriptor** (const VSW_KIND &eV, const REGULARITY_KIND &eR)
- static std::string **fieldClassDescriptor2** (const VSW_KIND eVSW1, const REGULARITY_KIND eReg1, const VSW_KIND eVSW2, const REGULARITY_KIND eReg2)
- static size_type **multipoleIndex** (const long l, const long m)
- static void **inverseMultipoleIndex** (const size_type index, long &l, long &m)
- static size_type **multipoleIndex** (long l, long m, size_t l_max)
- static void **inverseMultipoleIndex** (size_t index, size_t l_max, long &l, long &m)
- static size_type **fieldClassIndex** (const VSW_KIND eVSW, const REGULARITY_KIND eReg)
- static size_type **VSW_index** (const VSW_KIND eVSW)
- static void **inverse_VSW_index** (const size_type index, VSW_KIND &eVSW)
- static size_type **fieldClassIndex2** (const VSW_KIND eVSW1, const REGULARITY_KIND eReg1, const VSW_KIND eVSW2, const REGULARITY_KIND eReg2)
- static void **inverseFieldClassIndex** (const size_type index, VSW_KIND &eVSW, REGULARITY_KIND &eReg)

- static void **inverseFieldClassIndex2** (const size_type &index, VSW_-KIND &eVSW1, REGULARITY_KIND &eReg1, VSW_KIND &eVSW2, REGULARITY_KIND &eReg2)
- static size_type **rawFieldClassIndex2** (const size_type &nBlockRow, const size_type &nBlockCol)
- static void **inverseRawFieldClassIndex2** (const size_type &index, size_-type &nBlockRow, size_type &nBlockCol)
- static size_type **fieldClassDimensions** (void)
- static size_type **fieldClassDimensions2** (void)
- static void **blockIndexVectors** (const std::pair< REGULARITY_KIND, REGULARITY_KIND > &reg12_1, std::vector< long > &vnRows, std::vector< long > &vnCols)
- static void **blockIndexVectors** (const std::pair< REGULARITY_KIND, REGULARITY_KIND > &reg12_1, const std::pair< REGULARITY_-KIND, REGULARITY_KIND > &reg12_2, std::vector< long > &vnRows, std::vector< long > &vnCols)
- static void **blockIndexVectors** (const std::vector< std::pair< REGULARITY_KIND, REGULARITY_KIND > > &vReg12, std::vector< long > &vnRows, std::vector< long > &vnCols)
- static size_type **monolithicIndex** (long l, long m, size_t l_max)
- static void **inverseMonolithicIndex** (size_t index, size_t l_max, long &l, long &m)
- static size_type **monolithicIndex_l** (long l, size_t l_max)
- static void **inverseMonolithicIndex_l** (size_type index, size_t l_max, long &l)

- static size_type **monolithicIndex** (long l, long m, VSW_KIND eVSW, size_t l_max)
- static void **inverseMonolithicIndex** (size_t index, size_t l_max, long &l, long &m, VSW_KIND &eVSW)
- static size_type **monolithicIndex_l** (long l, VSW_KIND eVSW, size_t l_max)
- static void **inverseMonolithicIndex_l** (size_type index, size_t l_max, long &l, VSW_KIND &eVSW)
- static size_t **multipoleIndexMCentric** (long l, long m, size_t l_max)
- static void **inverseMultipoleIndexMCentric** (size_t n, size_t l_max, long &l, long &m)
- static size_type **monolithicIndexMCentric** (long l, long m, size_t l_max)
- static void **inverseMonolithicIndexMCentric** (size_t index, size_t l_max, long &l, long &m)
- static size_type **monolithicIndexMCentric** (long l, long m, VSW_KIND eVSW, size_t l_max)
- static void **inverseMonolithicIndexMCentric** (size_t index, size_t l_max, long &l, long &m, VSW_KIND &eVSW)

1.7.170 TMatrix::vectorFieldKet< T > Class Template Reference

Inheritance diagram for TMatrix::vectorFieldKet< T >:



Public Types

- typedef size_t **size_type**
- typedef numberTraits< T >::magnitudeType **R**
- typedef indefiniteVector< T > **fieldClassLayerVectorType**
- typedef gmm::row_matrix< fieldClassLayerVectorType > **ket_data_type**

Public Member Functions

- **vectorFieldKet** (const vectorFieldKet &other)
- **vectorFieldKet** (const long &lMax)
- virtual fieldKetBase< T >::field_ket_kind **kind** (void) const
- virtual fieldKetBase< T > * **clone** (void) const
- long **lMax** (void) const
- void **nnzByRawIndex** (std::map< long, long > &mN) const
- void **getCommonIndexVector** (std::vector< long > &vn) const throw ()
- void **toCompressedFormat** (blockVector< T > &vBlocks, std::vector< long > &vN) const throw ()
- void **fromCompressedFormat** (const blockVector< T > &vBlocks, const std::vector< long > &vN) throw ()
- void **toDenseFormat** (std::vector< T > &dest) const throw ()

- void **fromDenseFormat** (const std::vector< T > &src, REGULARITY_-KIND eReg) throw ()
- const fieldClassLayerVectorType & **operator()** (const VSW_KIND &eVSW1, const REGULARITY_KIND &eReg1) const
- fieldClassLayerVectorType & **operator()** (const VSW_KIND &eVSW1, const REGULARITY_KIND &eReg1)
- gmm::linalg_traits< ket_data_type >::const_sub_row_type::reference **operator()** (const VSW_KIND &eVSW, const REGULARITY_KIND &eReg, const long &l, const long &m) const
- gmm::linalg_traits< ket_data_type >::sub_row_type::reference **operator()** (const VSW_KIND &eVSW, const REGULARITY_KIND &eReg, const long &l, const long &m)
- vectorFieldKet & **operator+=** (const vectorFieldKet &other)
- vectorFieldKet & **operator-=** (const vectorFieldKet &other)
- vectorFieldKet & **operator*=** (const T &factor)
- vectorFieldKet & **applyCurl** (void)
- void **clear** (void)
- void **resize** (const long &lMax)
- void **clean** (const T &eps)
- void **clear** (VSW_KIND eVSW, REGULARITY_KIND eReg)
- void **clear** (VSW_KIND eVSW)
- void **clear** (REGULARITY_KIND eReg)
- void **fill** (const T &constant)
- void **apply** (const T &x, const T &phi, const std::vector< T > &vU, std::vector< T > &vEr, std::vector< T > &vEt, std::vector< T > &vEp,

std::vector< T > &vBr, std::vector< T > &vBt, std::vector< T > &vBp, const T &k=one< T >()) const throw ()

- template<class V >
  void **apply** (const V &vX, const std::vector< T > &vU, const std::vector< T > &vP, const std::vector< unsigned long > &vnXX, const std::vector< unsigned long > &vnUU, const std::vector< unsigned long > &vnPP, std::vector< T > &vEr, std::vector< T > &vEt, std::vector< T > &vEp, std::vector< T > &vBr, std::vector< T > &vBt, std::vector< T > &vBp, const T &k=one< T >()) const throw ()
- void **nnzByIndex** (std::map< long, long > &mL, std::map< long, long > &mM) const
- template<class U >
  void **magnitudeVectors** (std::vector< U > &vVSW_M, std::vector< U > &vVSW_N, const long &nType=0) const throw ()
- void **coefficientVectors** (std::vector< T > &vVSW_M, std::vector< T > &vVSW_N, const long &m_, REGULARITY_KIND eReg) const throw ()
- void **write** (std::ostream &os, bool writeIndices=false) const
- void **writeSignature** (std::ostream &os) const
- virtual bool **writeBinary** (abstractCommHandle ∗fp) const
- virtual bool **readBinary** (abstractCommHandle ∗fp)

Static Public Member Functions

- static void **getDenseIndexVector** (long l_max, std::vector< long > &vn) throw ()

Protected Member Functions

- void **calcVSWIncrement** (const long &l_, const long &m_, const VSW_-KIND eVSW, const T &factor, const std::vector< T > &vjl, const std::vector< T > &vDjl, const gmm::dense_matrix< T > &aPlm, const gmm::dense_matrix< T > &aDPlm, const T &exp_imp, std::vector< T > &vEr, std::vector< T > &vEt, std::vector< T > &vEp, bool bMagneticInduction) const
- void **calcVSWIncrement** (const long &l_, const long &m_, const VSW_-KIND eVSW, const T &factor, const gmm::dense_matrix< T > &ajl, const gmm::dense_matrix< T > &aDjl, const gmm::dense_matrix< T > &aPlm, const gmm::dense_matrix< T > &aDPlm, const std::vector< T > &exp_imp, const std::vector< unsigned long > &vnXX, const std::vector< unsigned long > &vnUU, const std::vector< unsigned long > &vnPP, std::vector< T > &vEr, std::vector< T > &vEt, std::vector< T > &vEp, bool bMagneticInduction) const
- void **mulOrdinateFactors** (const T &X, const T &U, std::vector< T > &vEr, std::vector< T > &vEt, std::vector< T > &vEp, const T &k) const
- void **mulOrdinateFactors** (const std::vector< T > &vX, const T &U, std::vector< T > &vEr, std::vector< T > &vEt, std::vector< T > &vEp, const T &k) const
- void **mulOrdinateFactors** (const T &X, const std::vector< T > &vU, std::vector< T > &vEr, std::vector< T > &vEt, std::vector< T > &vEp, const T &k) const
- void **mulOrdinateFactors** (const std::vector< T > &vX, const std::vector< T > &vU, std::vector< T > &vEr, std::vector< T > &vEt, std::vector< T >

&vEp, const T &k) const

- template<class V >
  void **mulOrdinateFactors** (const V &vX, const std::vector< T > &vU, const std::vector< unsigned long > &vnXX, const std::vector< unsigned long > &vnUU, std::vector< T > &vEr, std::vector< T > &vEt, std::vector< T > &vEp, const T &k) const

Protected Attributes

- ket_data_type **ketData_**

Friends

- void **tmatrix** (const vectorFieldKet< T > &src, vectorFieldKet< T > &dest) const

template<class T> class TMatrix::vectorFieldKet< T >

References

[1] F. E. Wagner, S. Haslbeck, and L. Stievano et al. Before striking gold in gold-ruby glass. *Nature*, 407(6805):691–692, Oct. 2000. 1

[2] H. C. van de Hulst. *Light Scattering by Small Particles*. Wiley, NY, 1957. 1, 29, 44

[3] G. Mie. Beiträge zur Optik trüber Medien, speziell kolloidaler Metallösungen. *Annalen der Physik*, 25(3):377–445, 1908. 1, 18

[4] P. Debye. Der Lichtdruck auf Kugeln von beliebigem Material. *Annalen der Physik*, 30:57–136, 1909. 1, 22, 31, 118

[5] S. L. McCall and P. M. Platzman. Surface Enhanced Raman Scattering. *Phys. Lett. A*, 77(5):381–383, June 1980. 1

[6] P. K. Aravind and H. Metiu. The Enhancement of Raman and Fluorescent Intensity by Small Surface Roughness, Changes in Dipole Emission. *Chem. Phys. Lett.*, 74(2):301–305, Sept. 1980. 1

[7] P. K. Aravind, A. Nitzan, and H. Metiu. The Interaction Between Electromagnetic Resonances and its Role in Spectroscopic Studies of Molecules Adsorbed on Colloidal Particles or Metal Spheres. *Surf. Sci.*, 110:189–204, 1981. 1, 2

[8] W. Faulk and G. Taylor. An Immunocolloid Method for the Electron Microscope. *Immunochemistry*, 8:1081–1083, 1979. 2

[9] E. Romano, C. Stolinsky, and N. Hugh-Jones. Ultrastructural Localisation of Intracellular Antigens by the use of Protein A-Gold Complex. *Immunochemistry*, 11:521–522, 1974. 2

[10] J. Roth and M. Bendayan amd L. Orci. FITC-Protein A-Gold Complex for Light and Electron Microscopic Immunocytochemistry. *J. Histochem. Cytochem.*, 28:55–57, 1980. 2

[11] E. De Harven, R. Leung, and H. Christensen. A Novel Approach for Scanning Electron Microscopy of Colloidal Gold Labelled Cell Surfaces. *J. Cell Biol.*, 99:53–57, 1984. 2

[12] L. C. Hoyer, J. C. Lee, and C. Bucana. Scanning Immunoelectron Microscopy for the Identification and Mapping of Two or More Antigens on Cell Surfaces. *Scanning Electron Microscopy 1979*, 3:629–636, 1979. 2

[13] L. Scopsi, L. I. Larsson, L. Bastholm, and M. H. Nielsen. Silver Enhanced Colloidal Gold Probes as Markers for Scanning Electron Microscopy. *Histochemistry*, 86:35–41, 1986. 2

[14] I. R. Nabiev, H. Morjani, and M. Manfait. Selective analysis of antitumor drug interaction with living cancer cells as probed by surface-enhanced Raman spectroscopy. *Euro. Biophys. J.*, 19(6):311–316, 1991. 2

[15] K. Kneipp et al. Surface-Enhanced Raman Spectroscopy in Single Living Cells Using Gold Nanoparticles. *App. Spect.*, 56(2):150–154, 2002. 2

[16] S. Nie and S. R. Emory. Probing single molecules and single nanoparticles by surface-enhanced Raman scattering. *Science*, 275(5303):1102–1106, 1997. 2

[17] K. Sokolov, G. Chumanov, and T. M. Cotton. Ehancement of Molecular Fluorescence near the Surface of Colloidal Metal Films. *Anal. Chem.*, 70(18):3898–3905, Sept. 1998. 2

[18] G. Chumanov, K. Sokolov, B. W. Gregory, and T. M. Cotton. Colloidal Metal Films as a Substrate for Surface-Enhanced Spectroscopy. *J. Phys. Chem.*, 99:9466–9471, 1995. 2

[19] R. Elghanian et al. Selective colorimetric detection of polynucleotides based on the distance-dependent optical properties of gold nanoparticles. *Science*, 277(5329):1078–1080, 1997. 2

[20] R. Yasuda et al. Resolution of distinct rotational substeps by submillisecond kinetic analysis of F1-ATPase. *Nature*, 410(6831):898–904, 2001. 2

[21] S. Schultz et al. Silver Enhanced Colloidal Gold Probes as Markers for Scanning Electron Microscopy. *PNAS*, 97(3):996–1001, 2000. 2

[22] K. Sokolov. Real-time vital optical imaging of precancer using anti-epidermal growth factor receptor antibodies conjugated to gold nanoparticles. *Cancer Research*, 63(9):1999–2004, May 2003. 2, 39

[23] P. C. Waterman. Matrix formulation of electromagnetic scattering. *Proceedings of the I.E.E.E.*, 53:805–812, 1965. 3, 30, 63, 74

[24] P. Barber and C. Yeh. Scattering of electromagnetic waves by arbitrarily shaped dielectric bodies. *Applied Optics*, 14(12):2864–2872, Dec. 1975. 3, 31, 49, 118

[25] A. D. Kiselev, V. Yu Reshetnyak, and T. J. Sluckin. Light scattering by optically anisotropic scatterers: T-matrix theory for radial and uniform anisotropies. *Phys. Rev. E*, 65(056609), May 2002. 3

[26] M. I. Mishchenko, L. D. Travis, and D. W. Mackowski. Capabilities and Limitations of a Current FORTRAN Implementation of the T-Matrix Method for Randomly Oriented, Rotationally Symmetric Scatterers. *J. Quant. Spectrosc. Radiat. Transfer*, 60(3):309–324, 1998. 3, 4, 13, 32

[27] G. Gouesbet, B. Maheu, and G. Grehen. Light scattering from a sphere arbitrarily located in a Gaussian beam, using a Bromwich formulation. *J. Opt. Soc. Am. A*, 5(9):1427–1443, Sept. 1988. 3

[28] G. Grehan, B. Maheu, and G. Gouesbet. Scattering of laser beams by Mie scatter centers: numerical results using a localized approximation. *Applied Optics*, 25(19):3539–3548, Oct. 1986. 3

[29] U. Kreibig. Optical Properties of Small Particles in Insulating Matrices. In J. Davenas and P. M. Rabette, editors, *Contribution of Clusters Physics to Material Science and Technology; From Isolated Clusters to Aggregated Materials*, pages 373–423. NATO Advanced Study Institute, Klewer Academic Publishers, 1986. 3, 11

[30] U. Kreibig and M. Vollmer. *Optical Properties of Metal Clusters*. Springer-Verlag, 1995. 3, 11, 42, 46

[31] J. D. Joannopoulos, R. D. Meade, and J. N. Winn. *Photonic Crystals: molding the flow of light*. Princeton University Press, 1995. 3

[32] U. Frisch. Wave Propagation in Random Media. In A. T. Bharucha-Reid, editor, *Probabilistic Methods in Applied Mathematics*, volume 1, pages 75–198. Academic Press, 1968. 3, 4, 42

[33] A. Ishimaru. *Multiple Scattering, Turbulence, Rough Surfaces, and Remote Sensing*, volume 2 of *Wave Propagation and Scattering in Random Media*. Academic Press, 1978. 3, 42

[34] F. Borghese, P. Denti, and R. Saija et al. Multiple Electromagnetic Scattering from a Cluster of Spheres. I. Theory. *Aerosol Science and Technology*, 3(2):227–235, 1984. 4, 67, 69, 106

[35] O. I. Sindoni, F. Borghese, and P. Denti et al. Multiple Electromagnetic Scattering from a Cluster of Spheres. II. Symmetrization. *Aerosol Science and Technology*, 3(2):237–243, 1984. 4

[36] P. Debye and A. M. Burche. Scattering by an Inhomogenous Solid. *J. Appl. Phys.*, 20:518–525, June 1949. 4

[37] P. Debye, H. R. Anderson Jr., and H. Brumberger. Scattering by an Inhomogenous Solid. II. The Correlation Function and Its Application. *J. Appl. Phys.*, 28(6):679–683, June 1957. 4, 55

[38] M. Tateiba. A new approach to the problem of wave scattering by many particles. *Radio Science*, 22(6):881–884, Nov. 1987. 4, 53, 55

[39] M. Tateiba, Y. Nanbu, and T. Oe. Numerical Analysis of the Effective Dielectric Constant of the Medium where Dielectric Spheres are Randomly Distributed. *IEICE Trans. Electron.*, E76-C(10):1461–1467, Oct. 1993. 4, 53

[40] M. Tateiba and Y. Nanbu. Condition for random distribution of many dielectric spheres to be random for a coherent field. *Radio Science*, 28(6):1203–1210, Nov.-Dec. 1993. 4, 53

[41] Y. Nanbu and M. Tateiba. The Effective Permittivity of the Discrete Random Medium Composed of Dielectric Spheres and Homogenous Space in the Region of Low to Resonance Frequencies. *IEEE AP-S Int. Symp.*, 3:2354–2357, 1994. 4, 53

[42] M. Tateiba. Electromagnetic Wave Scattering in Media Whose Particles are Randomly Displaced from a Uniformly Ordered Spatial Distribution. *IEICE Trans. Electron.*, E78-C(10):1357–1365, Oct. 1995. 4, 53

[43] L. D. Landau, E. M. Lifshitz, and L. P. Pitaevskii. *Electrodynamics of Continuous Media*, volume 8 of *Landau and Lifshitz Course of Theoretical Physics*. Pergamon Press, 2nd edition, 1984. 7, 8

[44] J. D. Jackson. *Classical Electrodynamics*. John Wiley & Sons, 3rd edition, 1999. 8, 9, 10, 15, 16, 18, 50

[45] P. Y. Yu and M. Cardona. *Fundamentals of semiconductors : physics and materials properties*. Springer, 3rd edition, 2001. 9, 10, 11

[46] P. B. Johnson and R. W. Christy. Optical Constants of the Noble Metals. *Phys. Rev. B*, 6(12):4370–4379, Dec. 1972. 11, 13

[47] G. W. Milton and D. J. Eyre. Finite Frequency Range Kramers Kronig Relations: Bounds on the Dispersion. *Phys. Rev. Lett.*, 79(16):3062–3065, Oct. 1997. 12

[48] Sandia, Los Alamos, Air Force Weapons Laboratory Technical Exchange Committee (SLATEC)

SLATEC is distributed by netlib (see http://www.netlib.org/) and also by:
Energy Science and Technology Software Center
P.O. Box 1020
Oak Ridge, TN 37831
USA. 14

[49] L. Tsang, J. A. Kong, and R. T. Shin. *Theory of Microwave Remote Sensing*. John Wiley & Sons, 1985. 15, 18, 32, 54, 72, 97

[50] P. M. Morse and H. Feshbach. *Methods of Theoretical Physics*, volume 2. McGraw-Hill, 1953. 15, 16

[51] M. E. Rose. *Multipole Fields*. John Wiley & Sons, 1955. 15, 68, 91

[52] J. A. Stratton. *Electromagnetic Theory*. McGraw-Hill, 1941. 18

[53] A. L. Aden and M. Kerker. Scattering of Electromagnetic Waves from Two Concentric Spheres. *J. Appl. Phys.*, 22(10):1242–1246, Oct. 1951. 18, 35

[54] G. Arfken. *Mathematical Methods for Physicists*. Academic Press, 3rd edition, 1985. 23

[55] W.J. Wiscombe. Improved Mie scattering algorithms. *Applied Optics*, 19(9):1505–1509, 1980. 23

[56] M.I. Mishchenko. Light scattering by randomly oriented axially symmetric particles. *JOSA A*, 8(6):871–882, 1991. 32

[57] N. G. Khlebtsov. Orientational averaging of light-scattering observables in the t-matrix approach. *Applied Optics*, 31(25):5359–5365, 1992. 32

[58] R. Ruppin. Optical Properties of small metal spheres. *Phys. Rev. B*, 11(8):2871–2876, Apr. 1975. 33, 34

[59] A. R. Melnyk and M. J. Harrison. Theory of Optical Excitation of Plasmons in Metals. *Phys. Rev. B*, 2(4):835–850, Aug. 1970. 33

[60] N. W. Aschroft and N. D. Mermon. *Solid State Physics*. W.B.Saunders, 1976. 34, 48

[61] J. Sinzig, U. Radtke, M. Quinten, and U. Kreibig. Binary Clusters - Homogeneous Alloys and Nucleus-Shell Structures. *Z. Physik D*, 26(1-4):242–245, May 1993. 36

[62] K. Gaus, E. Gratton, and E. P. W. Kable et al. Visualizing lipid structure and raft domains in living cells with two-photon microscopy. *PNAS*, 100(26):15554–15559, Dec. 2003. 38

[63] E. A. Miljan, E. J. Meuillet, and B. Mania-Fahrnell et al. Interaction of the Extracellular Domain of the Epidermal Growth Factor Receptor with Gangliosides. *J. Biol. Chem.*, 277(12):10108–10113, Mar. 2002. 38

[64] E. Odintsova, J. Voortman, and E. Gilbert et al. Tetraspanin CD82 regulates compartmentalisation and ligand-induced dimerization of EGFR. *J. Cell Science*, 116(22):4557–4566, 2003. 38

[65] J. S. Aaron, N. Niton, and K. Travis et al. Plasmon resonance coupling of metal nanoparticles for molecular imaging of carcinogenesis in vivo. *J. Biomed. Opt.*, 12(3):034007, 2007. 39, 45

[66] K. V. Sokolov, J. S. Aaron, and E. Hsu et al. Optical systems for in vivo molecular imaging of cancer. *Technology in Cancer Research & Treatment*, 2(6):491–504, 2003. 39

[67] C. S. Johnson and D. A. Gabriel. *Laser Light Scattering*. Dover, 1995. 45

[68] B. Chu. *Laser Light Scattering*. Academic Press, 1974. 45

[69] S. N. Timasheff and R. Townsend. Light Scattering. In S. J. Leach, editor, *Physical principles and techniques of protein chemistry*, volume B, page 147. Academic Press, 1970. 45

[70] E. P. Pittz, J. C. Lee, and B. Bablouzian et al. *Methods Enzymol.*, 27:209–256, 1971. 45

[71] K. Travis. Optical Scattering from Nanoparticle Aggregates. Master's thesis, University of Texas at Austin, 2004. 46

[72] P. C. Waterman and N. E. Pedersen. Electromagnetic scattering by periodic arrays of particles. *Journal of Applied Physics*, 59(8):2609–2618, April 1986. 48, 56

[73] C. Mätzler. Autocorrelation functions of granular media with free arrangement of spheres, spherical shells or ellipsoids. *J. Appl. Phys.*, 81(3):1509–1517, Feb. 1997. 55

[74] D.W. Marquardt. An Algorithm for Least-Squares Estimation of Nonlinear Parameters. *SIAM Journal on Applied Mathematics*, 11(2):431–441, June 1963. 58

[75] M.I. Mishchenko, G. Videen, V.A. Babenko, N.G. Khlebtsov, and T. Wriedt. T-matrix theory of electromagnetic scattering by particles and its applications: a comprehensive reference database. *J. Quant. Spectrosc. Radiat. Transfer*, 88:357–406, 2004. 62

[76] K.A. Fuller and G.W. Kattawar. Consummate solution to the problem of classical electromagnetic scattering by an ensemble of spheres. ii.clusters of arbitrary configuration. *Optics Letters*, 13:1063–1065, 1988. 64

[77] S. V . Mytnichenko. Distorted-wave Born approximation in the case of an optical scattering potential. *Physica B*, 355:244–249, 2005. 64

[78] D. W. Mackowski. Analysis of Radiative Scattering for Multiple Sphere Configurations. *Proc. R. Soc. Lond. A*, 433(1889):599–614, 1991. 64, 67, 69, 106

[79] D. W. Mackowski and M. I. Mishchenko. Calculation of the T matrix and the scattering matrix for ensembles of spheres. *J. Opt. Sci. Am. A*, 13(11):2266–2278, 1996. 65, 74, 78

[80] S. Stein. Addition theorems for spherical wave functions. *Q. appl. Math.*, 19:15–24, 1961. 67

[81] O.R. Cruzan. Translational addition theorems for spherical vector wave functions. *Q. appl. Math.*, 20:33–39, 1962. 67

[82] W.C. Chew. *J. Electron Waves Applic.*, 6:133, 1992. 67

[83] Brian Stout, Jean-Claud Auger, and Jacques Lafait. A transfer matrix approach to local field calculations in multiple-scattering problems. *Journal of Modern Optics*, 49(13):2129–2152, 2002. 69, 78, 109

[84] Brian Stout, Jean-Claud Auger, and Jacques Lafait. Individual and aggregate scattering matrices and cross-sections: conservation laws and reciprocity. *Journal of Modern Optics*, 48(14):2105–2128, 2001. 76, 78

[85] P.R. Siqueira and K. Sarabandi. *IEEE Trans. Antennas Propagation*, 48:317, 2000. 78

[86] GCC, the GNU Compiler Collection.
http://gcc.gnu.org/.
Free Software Foundation, Inc.
59 Temple Place - Ste. 330
Boston, MA 02111
USA. 84, 100

[87] INTEL. Intel C++ Compiler 11.1. Software and reference material available at http://software.intel.com/en-us/intel-compilers/. 84, 100

[88] The Portland Group. PGI C++ Compiler 7.2. Software and reference material available at http://www.pgroup.com/. 84

[89] The GNU Multiple Precision Arithmetic Library.
http://gmplib.org/. 84, 85

[90] Guillaume Hanrot, Vincent Lefèvre, Patrick Pélissier, Philippe Théveny, and Paul Zimmermann. The MPFR Library.
http://www.mpfr.org/. 84, 85

[91] Yves Renard. Gmm++ Library.
http://download.gna.org/getfem/html/homepage/gmm.html.
84, 98, 107

[92] The Trilinos Project.
http://trilinos.sandia.gov/.
Sandia National Laboratories; USA. 84, 99, 112

[93] Matteo Frigo and Steven G. Johnson. Fastest Fourier Transform in the West (FFTW).
http://www.fftw.org/. 84

[94] D. Healy Jr., D. Rockmore, P. Kostelec, and S. Moore. FFTs for the 2-Sphere - Improvements and Variations. *The Journal of Fourier Analysis and Applications*, 9(4):341–385, 2003. 84, 91

[95] Victor Shoup. NTL: A Library for doing Number Theory.
http://www.shoup.net/ntl/.
New York University
Courant Institute
251 Mercer Street
New York, NY 10012
USA. 84

[96] Bruno Haible. CLN - Class Library for Numbers.
http://www.ginac.de/CLN/. 84

[97] Milton Abramowitz and Irene A. Stegun. *Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables*. Dover, New York, ninth dover printing, tenth gpo printing edition, 1964. 86, 89, 91, 92

[98] LTI-Lib Computer Vision Library.
http://ltilib.sourceforge.net/doc/homepage/index.shtml.
Lehrstuhl für Technische Informatik

RWTH Aachen University
D-52062 Aachen, Deutschland. 87

[99] A. R. Barnett. An Algorithm for Regular and Irregular Coulomb and Bessel Functions of Real Order to Machine Accuracy. *Computer Physics Communications*, 21:297–314, 1981. 89

[100] J. J. Sakarai. *Modern Quantum Mechanics*. Addison-Wesley, 1994. 92

[101] W. Gander and W. Gautschi. Adaptive Quadrature – Revisited. *BIT*, 40(1):84–101, 2000. 93

[102] Begnaud Francis Hildebrand. *Introduction to numerical analysis: 2nd edition*. Dover Publications, Inc., New York, NY, USA, 1987. 94

[103] J. Rasch and A. C. H. Yu. Efficient Storage Scheme for Precalculated Wigner 3J, 6J, and Gaunt Coefficients. *SIAM J. SCI. Comput.*, 25(4):1416–1428, 2003. 95, 96

[104] IEEE. IEEE Std. 1003.1c-1995 thread extensions, 1995. Formerly POSIX.4a. now included in 1003.1-1996. 99, 101

[105] OpenMP Architecture Review Board. OpenMP Application Program Interface. Specification, 2008. 99

[106] Robert D. Blumofe and Christopher F. Joerg and Bradley C. Kuszmaul and Charles E. Leiserson and Keith H. Randall and Yuli Zhou. Cilk: An Efficient Multithreaded Runtime System. In *Proceedings of the Fifth ACM SIGPLAN Symposium on Principles & Practice of Parallel Programming (PPOPP)*, pages 207–216, 1995. 100

[107] Charles E. Leiserson. The Cilk++ concurrency platform. In *Digital Arts and Culture (DAC)*, pages 522–527. ACM, 2009. 100

[108] Leslie Lamport. The mutual exclusion problem: part i—a theory of interprocess communication. *J. ACM*, 33(2):313–326, 1986. 101

[109] Leslie Lamport. The mutual exclusion problem: partii—statement and solutions. *J. ACM*, 33(2):327–348, 1986. 101

[110] Message Passing Interface Forum. MPI: A Message-Passing Interface Standard, Version 2.2.
`http://www.mpi-forum.org/`, 2009. 102

[111] S. Drew, K. J. Gouph, and J. Ledermann. Implementing Zero Overhead Exception Handling. Technical Report 95-12, Faculty of Information Technology, Queensland University of Technology, 1995. 115

[112] Kyle Wilson. Exceptions and Error Codes.
`http://gamearchitect.net/Articles/ExceptionsAndErrorCodes.html`. 115

[113] International Standards Organization. Draft Technical Report on C++ Library Extensions. International Standard ISO/IEC DTR 19768:2005. 115

[114] Boost C++ libraries.
`http://www.boost.org/`. 115

[115] J. S. Aaron, K. Travis, N. Harrison, and K. V. Sokolov. Dynamic Imaging of Molecular Assemblies in Live Cells Based on Nanoparticle Plasmon Resonance Coupling. *Nano Letters*, 9(10):3612–3618, 2009. 117

[116] Mario Hentschel, Michael Saliba, Ralf Vogelgesang, Harald Giessen, A. Paul Alivisatos, and Na Liu. Transition from isolated to collective modes in plasmonic oligomers. *Nano Letters*, 10(7):2721–2726, 2010. 120, 138, 151

[117] J. Britt Lassiter, Heidar Sobhani, Jonathan A. Fan, Janardan Kundu, Federico Capasso, Peter Nordlander, and Naomi J. Halas. Fano resonances in plasmonic nanoclusters: Geometrical and chemical tunability. *Nano Letters*, 10(8):3184–3189, 2010. 120, 151

[118] Jonathan A. Fan, Kui Bao, Chihhui Wu, Jiming Bao, Rizia Bardhan, Naomi J. Halas, Vinothan N. Manoharan, Gennady Shvets, Peter Nordlander, and Federico Capasso. Fano-like interference in self-assembled plasmonic quadrumer clusters. *Nano Letters*, 0(0), 0. 120, 151

[119] Prashant K. Jain, Wenyu Huang, and Mostafa A. El-Sayed. On the universal scaling behavior of the distance decay of plasmon coupling in metal nanoparticle pairs: A plasmon ruler equation. *Nano Letters*, 7(7):2080–2088, 2007. 120, 144, 151, 152, 159

[120] A. E. Miroshnichenko, S. Flach, and Y. S. Kivshar. Fano resonances in nanoscale structures. *Rev. Mod. Phys.*, 82(3):2257–2298, 2010. 136

[121] L. Rayleigh. On the dynamical theory of gratings. *Proc. R. Soc. London, Ser. A*, 79:399–416, 1907. 136

[122] U. Fano. Sullo spettro di assorbimento dei gas nobili presso il limite dello spettro d'arco. *Nuovo Cimento*, 12:154–161, 1935. 136

[123] U. Fano. Effects of configuration interaction on intensities and phase shifts. *Phys. Rev.*, 124:1866–1878, 1961. 136

[124] Yong S Joe, Arkady M Satanin, and Chang Sub Kim. Classical analogy of fano resonances. *Physica Scripta*, 74(2):259–266, 2006. 137

[125] Dimitri van Heesch. Doxygen, Generate documentation from source code. http://www.doxygen.org/. 164

Vita

Kort Alan Travis graduated from West High School in Madison, Wisconsin, in 1978. He then attended the University of Wisconsin at Madison, the University of Minnesota at Minneapolis, and the University of Washington at Seattle, from where he received a B.S. degree in Physics in 1999. He entered the University of Texas Graduate School in June of 2000, from where he received an M.A. degree in Physics in 2004, with research focused on the optical properties of plasmonic nanoparticle aggregates. From 2004 through 2006, he joined in research at the University of Leipzig, in Germany, focusing there on optical interactions with biological cells and tissue. Late in 2006, he returned to the Ph.D. program at the University of Texas, and to his previous research on nanoparticle aggregates, which is the subject of the present dissertation. Prior to completing his B.S. at the University of Washington, he had worked, both as an employee in industry and as founder and manager of his own independent company, as an optics research technician and analyst, a robotics specialist, and a hardware interface engineer.

Permanent address: 117 Alden Dr.
Madison, WI 53705

This dissertation was typeset with LaTeX[†] by the author.

[†]LaTeX is a document preparation system developed by Leslie Lamport as a special version of Donald Knuth's TeX Program.