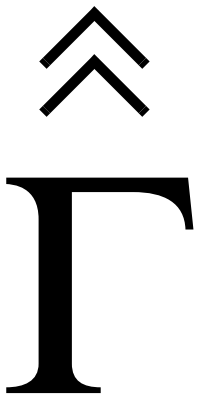
GAMMA

Level 1 Module



Author: Scott A. Smith

Date: March 14, 2000

1 Introduction

This module, Level 1, contains classes and functions which facilitate magnetic resonance simulations. These don't fall into any particular categories, they are grouped together only because they fit into GAMMA at the level of complexity. Much of the functionality supplied in this module is used in subsequent (higher level) GAMMA classes and modules.

2 Coordinates

2.1 Overview

The class ***coord*** *(coordinate)* is provided for the facile manipulation of points in 3-dimensional space. Each coordinate contains three real values or ordinates, *e.g.* {x,y,z}, {r,θ,ϕ}, or $\{u_1,u_2,u_3\}$. There is no specific internal distinction as to the coordinate system with which the coordinate is referenced. Each coordinate may represent a point in Cartesian space, a set of Euler angles, or anything which demands the simultaneous use of three numbers. This documentation will use {x,y,z} to indicate the three coordinate elements but keep in mind that coordinates need not be used strictly for the manipulation of Cartesian points.

Within the class are provided several general functions such as those for the I/O of these points. Functions are also provided to rotate, translate, and perform a variety of common coordinate transformations. In contrast to the generality of each coordinate, these functions typically assume that the points are associated with a specific type of coordinate system. For example, rotations in Cartesian space is different than rotations in Spherical coordinates.

See coordinate vector documentation for use of and manipulations on vectors of coordinates.

To use ***coord*** it is necessary to include the file ***Level1/coord.h*** or ***gamma.h***.

2.2 Available Functions

Scalar and Matrix Functions

2.3 Class Discussion

2.4 Figures

2.5 Example Programs

2.6 Basic Functions

2.6.1 coord

Usage:

```
#include <Level1/coord.h>
coord ( )
coord (double u1, double u2=0, double u3=0)
coord (coord &pt)
coord(ParameterSet& pset, idx=-1, warn=2);
```

Description:

The function ***coord*** is used to create a coordinate. This can be done by a simple declaration, by providing the three ordinate values, or by simply copying an existing coordinate. A fourth method is to read the coordinate in from a parameter set.

1. ***coord()*** - sets up an empty coordinate point which can later be explicitly specified.
2. ***coord(double u1, double u2, double u3)*** - sets up a coordinate which has components ***u1***, ***u2***, & ***u3***.
3. ***coord(coord &pt)*** - sets up a coordinate which is equivalent to the input coordinate, ***pt***.
4. ***coord(ParameterSet& pset, idx=-1, warn=2)*** - set up the coordinate from the parameter specifications in the parameter set ***pset***. The parameter searched for will have index ***idx*** (where -1 indicates no index). The flag ***warn*** sets what will occur should the coordinate parameters not be found in ***pset***. Default (2) causes a fatality, the value 1 causes output of non-fatal warnings and the coordinate set to the default coordinate, 0 causes no notice of a failure to set the coordinate.

Return Value:

Creates a new coordinate point.

Examples:

```
#include <gamma.h>
main()
  {
  coord pt;                          // Construct an empty coordinate point called pt.
  coord pt1(1,-3,7);                 // Construct coordinate pt1 with u1=1, u2=-3, u3=7.
  coord pt2(pt1);                    // Construct coordinate pt2, identical to pt1.
  ParameterSet pset;                 // A parameter set
  pset.read(String("filein.pset"));  // Read in parameters from an external file
  coord pt3(pset);                   // Read in coordinate from pset
  coord pt4(pset, 2, 1);             // Read in coordinate of index 2 from pset
  }
```

See Also: =

2.6.2 =

Usage:

```
#include <Level1/coord.h>
void operator = (coord &pt1)
```

Description:

This allows for the assignment of one coordinate from another.

Return Value:

Nothing, the function is void. The coordinate point given is altered.

Example:

```
#include <gamma.h>
main()
  {
  coord pt,pt1(5,6,7);                  // Define two coordinate points pt and pt1.
  pt = pt1;                             // Set pt equal to pt1.
  }
```

See Also:

2.7 I/O Functions

2.7.1 coord_getf

2.7.2 coord_setf

Usage:

```
#include <Level1/coord.h>
friend void coord_getf(int &type, int &science, int &digits, int &digits_after_dpoint)
friend void coord_setf(int type, int science, int digits, int digits_after_dpoint)
```

Description:

The function ***coord_getf*** returns the current output format for coordinates. The function ***coord_setf*** sets the output format of coordinates. All coordinates will be output according to the current settings until this function is re-invoked. The parameter meanings are shown in the following table.

Parameters Affecting The Output of Coordinates

Parameter	Affect on Output	Default
type	0 - Output ordinates in Cartesian format 1 - Output ordinates in Spherical format 2 - Output ordinates in Cylindrical format	0
science	!0 - Use scientific notation x.xxxey 0 - No scientific notation xxxx.xxx	0
digits	Total number of digits	6
digits_after_point	Digits after the decimal point[a]	2

a. A negative value is used to indicate no limit on the number of digits.

Example:

```
#include <gamma.h>
main()
  {
  coord pt(1,0,0);                  // Define a coordinate point.
  cout << "\n\t" << pt;             // Default print format: (1.00,   0.00,   0.00)
  coord_setf(1,0,8,3);              // Spherical output, R,θ,ϕ, no scientific notation, xxxx.xxx
  cout << "\n\t" << pt;             // Write under new print format: (     1.00,      1.57,      0.00);
  cout << "\n\n";
  }
```

Return Value:

Nothing. The parameters supplied as arguments are altered in coord_getf.

See Also: <<, print

2.7.3 <<

Usage:

#include <Level1/coord.h>
ostream& operator << (ostream& ostr, coord &pt)

Description:

The ***operator*** << is the standard output for printing a coordinate, ***pt***, by placement into the specified output stream, ***ostr***. The format of the output may be set with the function ***coord_setf***.

Example:

```
#include <gamma.h>
main()
  {
  coord pt;                          // Declare a coordinate
  cout << pt;                        // Print the coordinate to standard output.
  }
```

Return Value:

Returns a modified output stream.

See Also:

>>

2.7.4 >>

Usage:

#include <Level1/coord.h>
istream& operator >> (istream& istr, coord &pt)

Description:

The standard input ***operator*** >> reads the coordinate ***pt*** from the input stream ***istr***.

Example:

```
#include <gamma.h>
main()
  {
  complex pt;                        // Declare a coordinate
  cout << "\n\tPlease input z: ";    // Ask for the complex number
  cin >> z;                          // Read the complex number from standard input
  }
```

Return Value:

Returns a modified input stream.

See Also:

<<

2.8 Ordinate Access

2.8.1 get

Usage:

```
#include <Level1/coord.h>
double get(int i)
```

Description:

The function ***get*** returns the current value of the i^{th} ordinate. The value of ***i*** must be 0, 1, or 2.

Return Value:

A double.

Example:

```
#include <gamma.h>
main()
  {
  coord pt(5,6,7);                    // Define a coordinate point.
  double y = pt.get(1);               // Set y to the value 6.
  }
```

See Also: x, y, z

2.8.2 x

2.8.3 y

2.8.4 z

Usage:

```
#include <Level1/coord.h>
double coord::x ()
double coord::x (double u1)
double coord::y ()
double coord::y (double u2)
double coord::z ()
double coord::z (double u3)
```

Description:

The functions ***x, y***, and ***z*** are used to either obtain or set the first individual ordinates of a coordinate point. If no argument is given the function will return the requested ordinate whereas if a value is specified the oridinate will be set to that value. Although labeled ***x, y***, and ***z*** these deal with the 1st, 2nd, and 3rd ordinates respectively (and only x,y, & z in a Cartesian system).

Return Value:

A double precision number is returned.

Examples:

```
#include <gamma.h>
```

```
main()
  {
  coord pt(1,-3,7);                 // Construct coordinate pt {1, -3, 7}.
  cout << pt.x();                   // Write the first ordinate of pt, 1, to standard output.
  pt.x(3);                          // Set the first ordinate of pt to 3: {3,-3,7}.
  double tmp = 3 * pt.y();          // Set variable tmp to 3*-3, where -3 is 2nd ordinate.
  pt.y(1);                          // Set the second ordinate of pt to 1: {3, 1, 7}.
  double tmp = 5 + pt.z();          // Set temp to 5+7, 7 being the 3rd ordinate of pt
  pt.z(-1);                         // Set the 3rd ordinate of pt to -1: {3, 1, -1}
  }
```

See Also: get, xyz

2.8.5 xyz

Usage:

```
#include <Level1/coord.h>
void coord::xyz (double u1, double u2, double u3)
void coord::xyz (coord & pt)
```

Description:

The function ***xyz*** sets the values of the three ordinates of a coordinate point.

1. **coord.xyz(double u1, double u2, double u3)** - sets the ordinates to ***u1***, ***u2***, and ***u3*** respectively.
2. ***coord.xyz(coord &pt)*** - sets the components of the coordinate equal to those of coordinate ***pt***.

Return Value:

None, the function is void.

Examples:

```
#include <gamma.h>
main()
  {
  coord pt, pt1;                    // Create two empty points, pt and pt1.
  pt.xyz(-11,0,7);                  // Set pt coordinates {-11, 0, 7}.
  pt1.xyz(pt);                      // Set pt1 coordinates equal to those of pt {-11, 0, 7}.
  }
```

See Also: x, y, z, coord, =

2.9 Coordinate Pairs

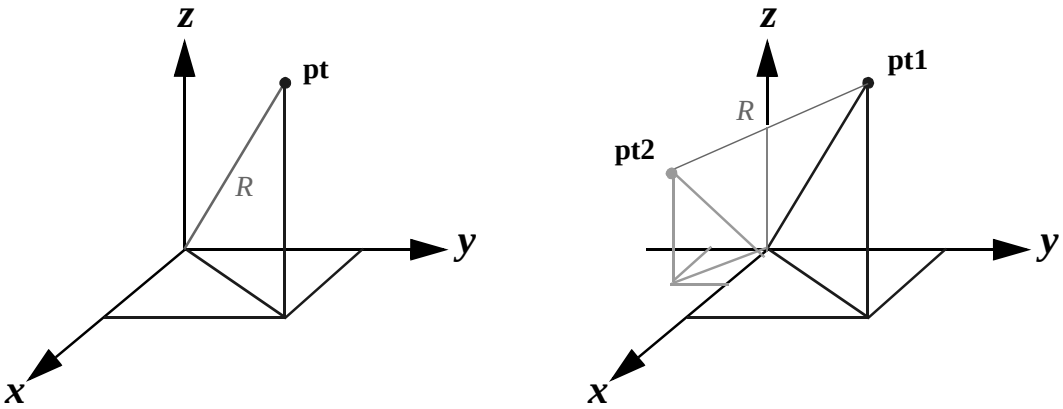
2.9.1 Rad

Usage:

```
#include <Level1/coord.h>
double coord::Rad ()
double Rad(double x, double y, double z)
double Rad(const coord &pt1, const coord &pt2)
```

Description:

Function ***Rad*** either returns or sets the value of the radius of a coordinate point, assuming that the point is Cartesian.



Return Value:

A double precision number is returned.

Examples:

```
#include <gamma.h>
main()
  {
  coord pt(-11,0,7);              // Create a coordinate pt at {-11, 0, 7}.
  cout << pt.radius();            // Radius to standard output, sqrt(11**2 + 7**2)
  cout << Rad(-11,0,7);           // Same result as above
  coord pt2(0,1,1);               // Another point at (0,1,1)
  cout << Rad(pt, pt2);           // Distance between two points, sqrt(11**2+1**2+6**2)
  }
```

Mathematical Basis:

When dealing with a single coordinate, ***Rad*** returns the value ***R*** as given in equation (2-1) on page 28, below left.

$$R = \sqrt{x^2 + y^2 + z^2} \qquad R = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2}$$

Obviously, this assumes the point is a Cartesian coordinate. In the case of two coordinates, the distance between them is determined from the equation on the right.

See Also: theta, phi

2.9.2 theta

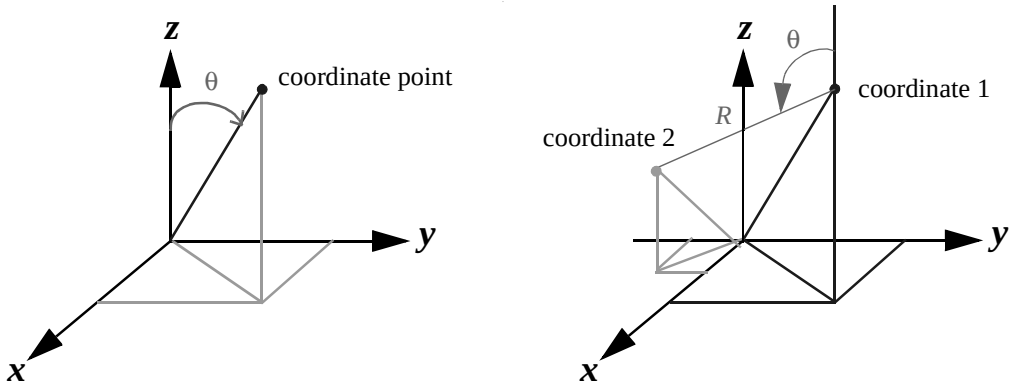
Usage:

```
#include <Level1/coord.h>
double coord::theta ()
double theta (double x, double y, double z)
double theta (coord &pt1, coord &pt2)
```

Description:

This function returns the value of the theta spherical component of a coordinate point.

Point & Point Pair Defined Theta Angle



Return Value:

A double precision number is returned.

Example(s):

```
#include <gamma.h>
main()
  {
  }
```

Mathematical Basis:

This function returns the value of ***theta*** in radians as given in equation (2-1) on page 28.

$$\theta = \operatorname{acos}(z/R)$$

Obviously this assumes the point is a Cartesian coordinate value. If the point were stored as {r,θ,ϕ} then the function y would be used to access this angle.

See Also: phi, Rad

2.9.3 phi

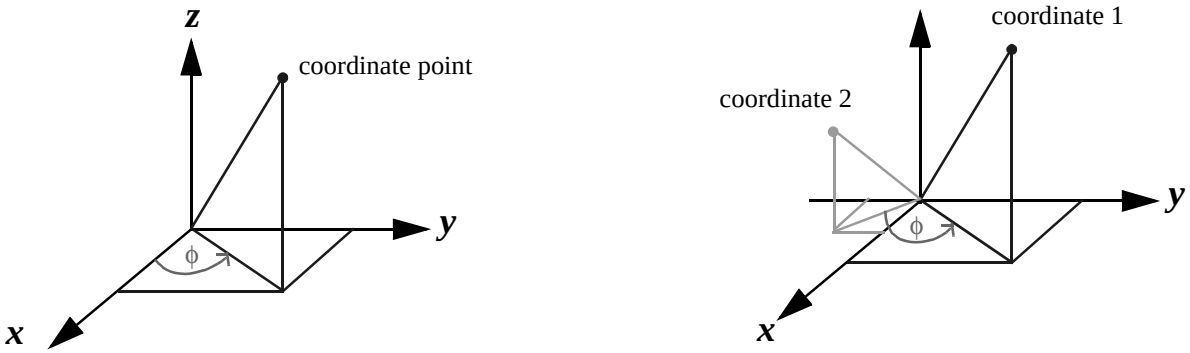
Usage:

#include <Level1/coord.h>
double coord::phi ()
double phi (double x, double y, double z)
double phi (coord &pt1, coord &pt2)

Description:

This function returns the value of the phi spherical component of a coordinate point.

Point & Point Pair Defined Phi Angle



Return Value:

A double precision number is returned.

Example(s):

```
#include <gamma.h>
main()
  {
  }
```

Mathematical Basis:

This function returns the value of the angle ***phi*** in radians as given in equation (2-1) on page 28.

$$\phi = \text{atan}(y/x)$$

Obviously this assumes the point is a Cartesian coordinate value. If the point were stored as {r,θ,ϕ} then the function z would be used to access the angle.

See Also: theta, Rad

2.9.4 invert

Usage:

#include <Level1/coord.h>
double coord::invert()

Description:

This function inverts all three ordinates of the point: ***{x,y,z} -> {1/x, 1/y, 1/z}***

Return Value:

Void, the coordinate is altered

Example(s):

```
#include <gamma.h>
main()
  {
  }
```

2.10 Rotations

2.10.1 Rz

2.10.2 Rx

2.10.3 Ry

Usage:

```
#include <Level1/coord.h>
static matrix coord::Rz(double phi, int rad=0)
static matrix coord::Rx(double theta, int rad=0)
static matrix coord::Ry(double beta, int rad=0)
```

Description:

The functions ***Rx***, ***Ry***, and ***Rz*** return matrices which perform rotations about the Cartesian coordinate axes. The returned arrays are 3x3 and meant to operate on Cartesian coordinates. The rotations follow the right-hand rule where the coordinate is rotated as the axes remain static. The input angles are in degrees unless the flag ***rad*** is set non-zero.

Return Value:

A 3x3 matrix is returned.

Example:

```
#include <gamma.h>
main()
  {
  }
```

See Also:xrotate, yrotate, zrotate

2.10.4 xrotate
2.10.5 yrotate
2.10.6 zrotate

Usage:

```
#include <Level1/coord.h>
coord coord::xrotate(double phi, int rad=0) const
coord coord::yrotate(double theta, int rad=0) const
coord coord::zrotate(double beta, int rad=0) const
```

Description:

The functions ***urotate*** return a coordinate rotated about the Cartesian axis ***u***. The rotations follow the right-hand rule where the coordinate is rotated as the axes remain static. The input angles are in degrees unless the flag ***rad*** is set non-zero.

Return Value:

A 3x3 matrix is returned.

Example(s):

```
#include <gamma.h>
main()
  {
  }
```

See Also: Rx, Ry, Rz

2.10.7 Ralpha
2.10.8 Rbeta
2.10.9 Rgamma

Usage:

```
#include <Level1/coord.h>
matrix coord::Rz(double phi, int rad) const
matrix coord::Rx(double theta, int rad) const
matrix coord::Ry(double beta, int rad) const
```

Description:

The functions ***Rx***, ***Ry***, and ***Rz*** return matrices which perform rotations about the Cartesian coordinate axes. The returned arrays are 3x3 and meant to operate on Cartesian coordinates. The rotations follow the right-hand rule where the coordinate is rotated as the axes remain static. The input angles are in degrees unless the flag ***rad*** is set non-zero.

Return Value:

A 3x3 matrix is returned.

Example:

```
#include <gamma.h>
main()
  {
```

```
}
```

See Also:xrotate, yrotate, zrotate

2.10.10 Rmx

Usage:

```
#include <Level1/coord.h>
matrix Rmx(double alpha, double beta, double gamma)
```

Description:

The function ***Rmx*** returns the matrix which rotates coordinates about an arbitrary axis as defined by the Euler angles alpha, beta, and gamma.

Return Value:

A 3x3 matrix is returned.

Example(s):

```
#include <gamma.h>
main()
  {
  }
```

See Also: Rmx1, Rmx2, Rmx3

2.10.11 rotate

Usage:

```
#include <Level1/coord.h>
coord coord::rotate (double alpha, double beta, double gamma)
coord coord::rotate (coord &pt)
coord coord::rotate_ip (double alpha, double beta, double gamma)
coord coord::rotate_ip (coord &pt)
```

Description:

Rotates the coordinate point by the Euler angles alpha, beta, and gamma. The angles are input in degrees.

Return Value:

A new coordinate point is returned.

Example(s):

```
#include <gamma.h>
main()
  {
  }
```

See Also:

2.11 Translations

2.11.1 trans_x

2.11.2 trans_y

2.11.3 trans_z

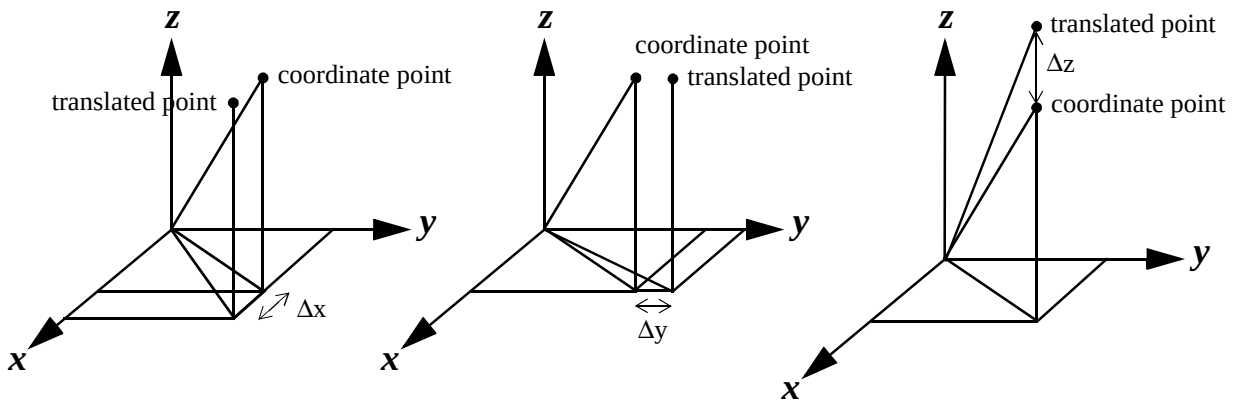
Usage:

```
#include <Level1/coord.h>
coord trans_x(double delx)
void trans_x_ip(double delx)
coord trans_y((double dely)
void trans_y_ip((double dely)
coord trans_z((double delz)
void trans_z_ip((double delz)
```

Description:

This set of functions will translate the coordinate point along a specific Cartesian axis by the value input. The function ***trans_u*** where ***u = {x,y,z}*** will translate the coordinate along the axis ***u***. The input value will be added to the appropriate ordinate. Functions with the suffix ***_ip*** do the same but will directly alter the coordinate rather than return the altered coordinate.

Coordinate Point Translations



Return Value:

Either the translated coordinate is returned, leaving the original point intact, or the point itself is translated and the function returns nothing.

Example(s):

```
#include <gamma.h>
main()
  {
  }
```

See Also: translate

2.11.4 translate

Usage:

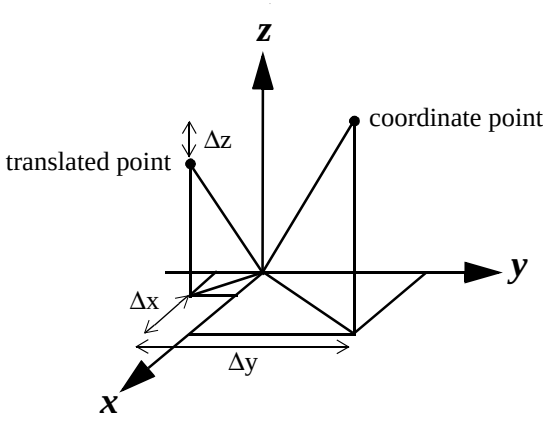
```
#include <Level1/coord.h>
coord translate(double delx, double dely, double delz)
coord translate(coord &del)
coord translate_ip(double delx, double dely, double delz)
coord translate_ip(coord &del)
```

Description:

This function translates the coordinate point along the three Cartesian axes by the values input.

1. translate(double delx, double dely, double delz) - copies then translates the coordinate along the three Cartesian axes by delx, dely, and delz.
2. translate(coord &del) - same as 1. but gets the delx, dely, and delz values from input coordinate del.
3. translate_ip(double delx, double dely, double delz) - same as 1. but alters the input coordinate.
4. translate_ip(coord &del) - same as 2. but alters the input coordinate.

Defined Point General Translation



Return Value:

Example(s):

```
#include <gamma.h>
main()
  {
  }
```

See Also: trans_x, trans_y, trans_z

2.12 Coordinates with Scalars and Matrices

2.12.1 *

2.12.2 *=

Usage:

```
#include <Level1/coord.h>
friend coord operator * (const coord &pt, double r)
friend coord operator *= (double r, const coord &pt )
friend void operator *= (coord &pt, double r )
```

Description:

These functions multiply all ordinates of point ***pt*** by the constant ***r***: ***{x,y,z} -> {rx, ry, rz}.***

Return Value: coordinate or void

Example(s):

```
#include <gamma.h>
main()
  {
  }
```

See Also: trans_x, trans_y, trans_z

2.12.3 /

2.12.4 /=

Usage:

```
#include <Level1/coord.h>
friend coord operator / (const coord &pt, double r)
friend void operator /= (coord &pt, double r )
```

Description:

These functions divide all ordinates of point ***pt*** by the constant ***r***: ***{x,y,z} -> {x/r,y/r, z/r}.***

Return Value: coordinate or void

Example(s):

```
#include <gamma.h>
main()
  {
  }
```

See Also: trans_x, trans_y, trans_z

2.12.5 *

Usage:

```
#include <Level1/coord.h>
friend coord operator / (const matrix &mxt, const coord& pt)
```

Description:

These functions multiply point ***pt*** by the matrix ***mx***: ***pt' = mx * pt .*** The input matrix must be 3x3

Return Value: coordinate

Example(s):

```
#include <gamma.h>
main()
  {
  }
```

See Also: trans_x, trans_y, trans_z

2.13 Coordinates with Parameters

2.13.1 param

Usage:

```
#include <Level1/coord.h>
SinglePar coord::param(const string& name) const
SinglePar coord::param(const string& name, constant string& state) const
```

Description:

Produces a GAMMA single parameter with ***name*** and optional statment ***state*** from the input coordinate. See documentation on class ***SinglePar***. This is used to build up parameter sets containing multiple coordinate (and other parameters) that may be written to an ASCII file in GAMMA parameter format (and subsequently reread).

Return : A single parameter

Example:

```
#include <gamma.h>
main()
  {
  coord pt(3.7,19.8);                 // Here is an coordinate
  SinglePar par = pt.param("pt");     // The parameter equivalent
  }
```

See Also: trans_x, trans_y, trans_z

2.13.2 read

Usage:

```
#include <Level1/coord.h>
int coord::read(const string& filename, int indx=-1, int warn=1)
int coord::read(const ParameterSet& pset, int indx=-1, int warn=1)
```

Description:

Reads a coordinate from either an ASCII file (in GAMMA parameter format) or from a parameter set. The coordinate is referenced by its parameter name which by default must be ***Coord***. If an index ***indx*** is specified (other than -1) then the coordinate name will be taken as ***Coord[indx]***. The integer ***warn*** indicates how to handle read failures: 0=nothing, 1=non-fatal warnings, 2=fatal error. The return of the function is true(1)/false(0).

Return: A single parameter

Example:

```
#include <gamma.h>
main()
  {
  coord pt;
  pt.read("coord.pset");                  // Try to read Coord from file
  ParameterSet pset("coords.pset");       // Read all parameters in file
  pt.read(pset, 3);                       // Try for Coord(3) from pset
  }
```

See Also: write

2.14 Description

Class ***coord*** allows for the facile manipulation of coordinate points. Each coordinate contains three real values, *e.g.* {x,y,z}, {r,θ,ϕ}, or $\{u_1,u_2,u_3\}$. Each coordinate may represent a point in Cartesian space, a set of Euler angles, or anything which demands the simultaneous use of three numbers. There is no specific internal distinction as to the coordinate system with which the coordinate is referenced. However many of the provided functions do assume that the coordinates are a specific type (e.g. the I/O functions).

2.14.1 Cartesian Coordinates

As Cartesian coordinates are the most common type of coordinates, there are several functions which deal specifically with this type. The mathematical constructs of the functions provided to deal with Cartesian coordinates will work properly only when the coordinate(s) are specified appropriately. The user must adhere to the definitions used within the provided functions.

Cartesian coordinates are viewed as existing in a right handed system as pictured below.

Right Handed Cartesian Coordinate System

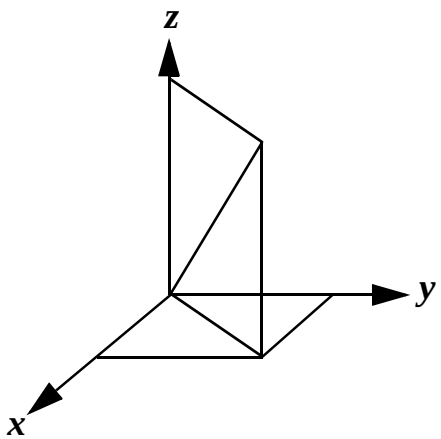


Figure A The Cartesian coordinate system used throughout GAMMA.

Functions are provided for obtaining the spherical and cylindrical coordinates of a Cartesian coordinate which adheres to this convention. The spherical and cylindrical coordinates are discussed in subsequent sections of this document.

2.14.2 Spherical Coordinates

Whenever spherical coordinates are related to Cartesian coordinates in GAMMA we adhere to use of the right hand coordinate system shown in the previous section as well as angle definitions

shown in the following figure.

Cartesian and Spherical Coordinate Systems

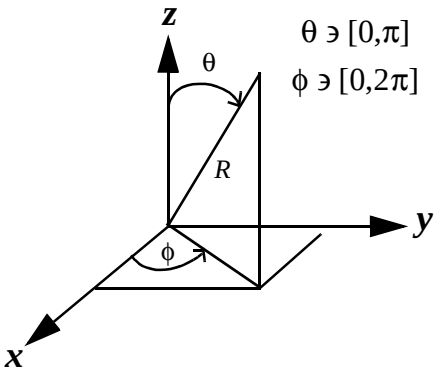


Figure B : The right handed Cartesian axes are shown as well as the spherical angles and radius.

As such, the following equations relate the two systems.

$$\begin{aligned} R &= \sqrt{x^2 + y^2 + z^2} & x &= R\sin\theta\cos\phi \\ \theta &= \operatorname{acos}(z/R) & y &= R\sin\theta\sin\phi \\ \phi &= \operatorname{atan}(y/x) & z &= R\cos\theta \end{aligned} \qquad (2\text{-}1)$$

Of course, because the inverse trigonometric functions always return their principle values, the equations for θ and ϕ cannot be used blindly.

2.14.3 Cylindrical Coordinates

Whenever cylindrical coordinates are related to Cartesian coordinates in GAMMA we adhere to use of the right hand coordinate system shown in an earlier section as well as angle definitions shown in the following figure.

Cartesian and Cylindrical Coordinate Systems

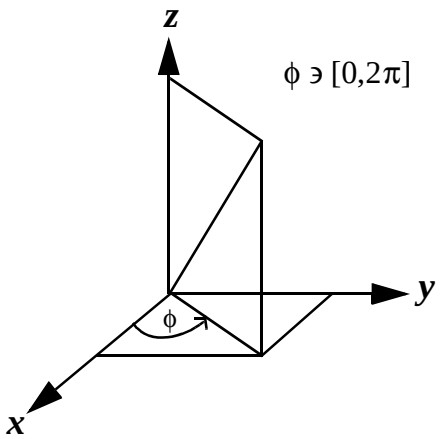


Figure C

This class uses these systems in functions which implicitly assume (or use) either Cartesian or spherical coordinates. As such, the following equations relate the two systems.

$$\begin{aligned} R &= \sqrt{x^2 + y^2} & x &= R\cos\phi \\ z &= z & y &= R\sin\phi \\ \phi &= \text{atan}(y/x) & z &= z \end{aligned} \tag{2-2}$$

2.14.4 Polar Space Coordinates

This class uses these systems in functions which implicitly assume (or use) either Cartesian or spherical coordinates. As such, the following equations relate the two systems.

Cartesian and Polar Space Coordinate Systems

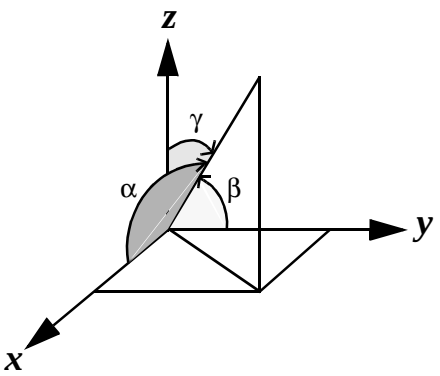


Figure D

This class uses these systems in functions which implicitly assume (or use) either Cartesian or spherical coordinates. As such, the following equations relate the two systems.

$$\begin{aligned} R &= \sqrt{x^2 + y^2 + z^2} & x &= R\cos\alpha \\ \alpha &= \text{acos}(x/R) & y &= R\cos\beta \\ \beta &= \text{acos}(y/R) & z &= R\cos\gamma \\ \gamma &= \text{acos}(z/R) \end{aligned} \tag{2-3}$$

know how the program views these functions depend and assume they use each coordinate may represent a point in Cartesian space, a set of Euler angles, or anything which demands the simultaneous use of three numbers. This documentation will use {x,y,z} to indicate the three coordinate elements but keep in mind that coordinates do not necessarily have to be used strictly for the manipulation of Cartesian coordinates.

Within the class are provided several general functions such as those for the I/O of these points. Functions are also provided to rotate, translate, and perform a variety of common coordinate transformations, but these typically assume that the points are associated with a specific type of coordinate system. For example, rotations in Cartesian space is different than rotations in Spherical

coordinates.

See class ***coord_vec*** for manipulation of a vector of coordinates. ***Coord_vec*** uses class ***coord*** for the vector points.

2.14.5 Coordinate Structure

The internal structure of class ***coord*** contains three doubles for the three ordinates (3-dimensional space) as well as 5 static[1] variables used to specify the output format.The contents are listed in the following Table, with their internal names.

Internal Structure of Class coord

Name	Description	Type	Name	Description	Type
cx	First Ordinate	double	_otype	Coordinate Type	static int
cy	Second Ordinate	double	_science	Scientific Notation	static int
cz	Third Ordinate	double	_digits	Total Digits Desired	static int
_form	Output Format	static String	_precise	Digits After Decimal	static int

Figure E Depiction of class coord contents, i.e. what each GAMMA defined coordinate contains.

The first four quantities define the coordinate: a point(3 ordinates), and the output format.

Structure of a Variable of Class coord

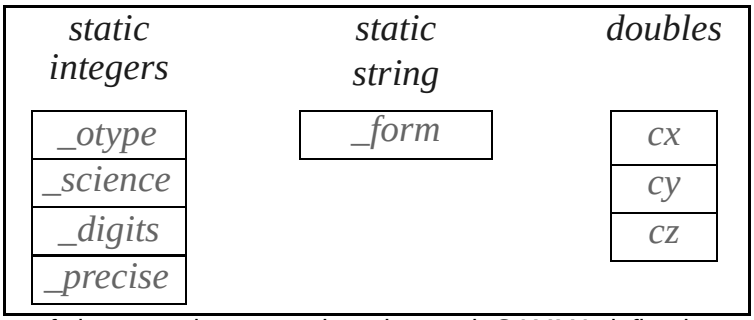


Figure F Depiction of class coord contents, i.e. what each GAMMA defined coordinate contains. However, note that the static variables are shared by all coordinates. Thus, for example, 10 coordinates will consist of 10x3 doubles yet only 4 integers and 1 string.

1. Static variables are declared only once per class. No matter how many coordinates are used in a program only one instance of the static variables are created. Thus, 1000 different coordinates will use 3000 doubles (ordinates) plus the five static variables (one string and 4 integers).

2.15 Parameters

This section describes how an ASCII file may be constructed that is self readable by a coordinate. The file can be created with any editor and is read with the coordinate member function ***read***.

Cartesian & Arbitrary Coordinates

Cartesian coordinates can be input either in Angstroms (default), nanometers, or meters depending upon the parameter name. GAMMA will internally scale the former two and store all in meters. Arbitrary values should be input in ***meters*** as these will NOT be scaled so that the user may apply his/her own scaling if necessary.

Cartesian *Coordinate Parameters*[a]

Parameter	Assumed Units	Examples Parameter (Type=3) : Value - Statement
Coord	A	Coord(0) (3) : (0.0, 0.0, 1.0) - Coordinate in Angstroms
Coordn	nm	Coordn(1) (3) : (2.0, 1.0, 1.0) - Coordinate in nanometers
Coordm	none, m	Coordm(2) (3) : (0.0, 0.0, 1.0) - Coordinate in meters

a. Parameters to set coordinates in a Cartesian mode. For general (unscaled) coordinates use Coordm.

Spherical Coordinates

Spherical coordinates can be input with the radius either in Angstroms (default), nanometers, or meters and with the angles in either degrees (default) or radians. Again, how the input values are interpreted depends upon the parameter name. GAMMA will internally store all of these coordinates in a Cartesian axis system with meter units

***Spherical Coordinate Parameters*[a]**

Parameter	Assumed Units	Examples Parameter (Type=3) : Value - Statement
CoordSph	A, deg	CoordSph(0) (3) : (1.0, 0.0, 0.0) - R is 1 Angstrom
CoordSphn	nm, deg	CoordSphn(1) (3) : (2.0, 90.0, 0.0) - R = 2 nm, θ=90
CoordSphm	m, deg	CoordSphm(2) (3) : (0.1, 0.0, 90.0) - R = 1m, ϕ=90
CoordSphR	A, rad	CoordSphR(0) (3) : (1.0, 0.0, 0.0) - R is 1 Angstrom
CoordSphRn	nm, rad	CoordSphRn(1) (3) : (2.0, 1.57, 0.0) - R = 2 nm, θ=180
CoordSphm	m, rad	CoordSphRm(2) (3) : (0.1, 0.0, 0.79) - R = 1m, ϕ=90

a. Shown are three possible parameters used to set a coordinates in a Spherical mode.

Simple Read of Coordinates From An ASCII File

file.asc

```
DI      (1) : 1.5      - Spin Quantum Number
DS      (1) : 0.5      - Spin Quantum Number
DCC     (1) : 70.3     - Dipolar Coupling (KHz)
Dtheta  (1) : 127.2    - Dipolar Orientation from PAS z (deg)
Dphi    (1) : 270.9    - Dipolar Orientation from PAS x(deg)
DI(2)       (1) : 0.5      - Spin Quantum Number
DS(2)       (1) : 0.5      - Spin Quantum Number
DCC(2)      (1) : 10.3     - Dipolar Coupling (KHz)
Dtheta(2)   (1) : 0.0      - Dipolar Orient. from PAS z (deg)
Dphi(2)     (1) : 270.9    - Dipolar Orient. from PAS x(deg)
```

code.cc

```
.............
.............
IntDip D;
D.read("file.asc");
IntDip D2;
D.read("file2.asc", 2);
.............
.
```

Figure G Specifying coordinates using an external ASCII file. Note that when angles are input in radians it is important to specify their values with good precision.

3 Coordinate Vectors

3.1 Overview

The class ***Coordinate Vector*** facilitates the manipulation of a list of coordinate points (class ***coord***). Each coordinate contains three values: x, y, and z in Cartesian space. Functions are provided to rotate, translate, and perform a variety of common coordinate transformations simultaneously over all coordinates in the vector. See class ***coord*** for information concerning specifics on individual coordinates

3.2 Available Functions

Individual Coordinate Access Functions

Parameter Set Functions

Coordinate Vector I/O Functions

3.3 Routines

3.3.1 coord_vec

Usage:

```
#include <Level1/coord_vec.h>
coord_vec ( )
coord_vec (int pts)
coord_vec (const coord_vec &cvec)
```

Description:

Function ***coord_vec*** is used to create a vector of coordinates.

1. coord_vec() - sets up an empty coordinate vector which can later be explicitly specified.
2. coord_vec(int pts) - sets up a coordinate vector which has ***pts*** coordinates (all 0,0,0);
3. coord_vec(const coord_vec &cvec) - new coordinate which is equivalent to the input coordinate, ***cvec***.

Return Value:

Creates a new coordinate vector.

Examples:

```
#include <gamma.h>
main()
  {
  coord_vec cvec;                  // Produces an empty coordinate vector.
  coord_vec cvec1(117);            // Produces coordinate vector with 117 coordinates.
  coord_vec cvec2(cvec1);          // Produces coordinate vector cvec2, equal to cvec1.
  }
```

See Also: =

3.3.2 =

Usage:

```
#include <Level1/coord_vec.h>
void operator = (coord_vec &cvec1)
```

Description:

This allows for the assignment of one coordinate vector to another.

Return: Void.

Example:

```
#include <gamma.h>
main()
  {
  coord_vec cvec, cvec1(1000);     // Define two coordinate vectors cvec and cvec1.
  cvec = cvec1;                    // Set coordinates of cvec equal to those in cvec1.
  }
```

3.4 Rotations

3.4.1 rotate

Usage:

```
#include <Level1/coord_vec.h>
coord_vec coord_vec::rotate(double alpha, double beta, double gamma)
coord_vec coord_vec::rotate(const coord &EA)
coord_vec coord_vec::rotate_ip(double alpha, double beta, double gamma)
coord_vec coord_vec::rotate_ip(const coord &EA)
coord_vec coord_vec::rotate(const matrix& Rmx) const
```

Description:

The function ***rotate*** allows the user to rotate the entire vector of coordinates with the Euler angles ***alpha***, ***beta***, and ***gamma***. These angle may be input as three double precision arguments or as a single coordinate ***EA***. The function returns the rotated vector. Functions ending with suffix ***_ip*** do the same rotation except they return void, rotating the vector directly. One additional function form will multiply all coordiantes by the 3x3 rotation matrix ***Rmx***.

Return Value:

Either the function returns a coordinate vector or it returns nothing (the input coordinate set is rotated in place).

Examples:

```
#include <gamma.h>
main()
  {
  coord_vec cvec, cvec1(1000);          // define two coordinate vectors cvec and cvec1.
  cvec = cvec1.rotate(0, 90, 45);       // cvec set equal to cvec1 rotated.
  cvec1.rotate_ip(130, 90, 15);         // cvec1 itself is rotated.
  coord EA(10, 20, 30);                 // set coordinate point with Euler angles 10, 20, & 30.
  cvec1.rotate_ip(EA);                  // cvec1 itself rotated by Euler angles in pt EA.
  }
```

See Also: xrotate, yrotate, zrotate

3.4.2 xrotate

3.4.3 yrotate

3.4.4 zrotate

Usage:

```
#include <Level1/coord_vec.h>
coord_vec coord_vec::xrotate(double theta, int rad=0) const
coord_vec coord_vec::yrotate(double theta, int rad=0) const
coord_vec coord_vec::zrotate(double phi, int rad=0) const
```

Description:

The functions ***urotate*** return a coordinate vector rotated about the Cartesian axis ***u***. The rotations follow the right-hand rule where all coordinates are rotated as the axes remain static. The input angles are in degrees unless the flag ***rad*** is set non-zero.

Return Value:

A coordinate vector is returned.

Examples:

```
#include <gamma.h>
main()
  {
  coord_vec cvec(2);
  coord_vec cvmy = cvec.xrotate(90.0);
  coord_vec cvx = cvec.yrotate(90.0);
  coord_vec cvy = cvmy.zrotate(180.0);
  }
```

3.5 Translations

3.5.1 translate

Usage:

```
#include <Level1/coord_vec.h>
coord_vec coord_vec::translate(double delx, double dely=0, double delz=0) const
coord_vec coord_vec::translate(const coord &del)
coord_vec coord_vec::translate_ip(double delx, double dely=0, double delz=0) const
coord_vec coord_vec::translate_ip(const coord &del)
```

Description:

The function ***translate*** allows the user to translate the entire vector of coordinates with the increments ***delx***, ***dely***, and ***delz***. The translation is given by

$$pt(i)_{translated} = \begin{Bmatrix} x(i)_{translated} \\ y(i)_{translated} \\ z(i)_{translated} \end{Bmatrix} = \begin{Bmatrix} x(i) + delx \\ y(i) + dely \\ z(i) + delz \end{Bmatrix} = pt(i) + \begin{Bmatrix} delx \\ dely \\ delz \end{Bmatrix}.$$

This makes sense as a translation only if the coordinates are Cartesian. The three ordinates, ***delx***, ***dely***, and ***delz***, can be input either individually or as a single coordinate. The functions with suffix ***_ip*** are perform the identical rotation except they act directly on the vector and return void.

Return Value:

Either the function returns a coordinate set (a copy of the input coordinate set but translated) or it returns nothing (the input coordinate set is translated in place).

Examples:

```
#include <gamma.h>
main()
  {
  coord_vec cvec, cvec1(1000);          // define two coordinate vectors cvec and cvec1.
  cvec = cvec1.translate(10, -4, 4.5);  // cvec set equal to cvec1 translated.
  cvec1.translate_ip(1.30, 9.0, -1.5);  // cvec1 itself is translated.
  coord del(1.0, 2.0, 3.0);             // set coordinate point with values 1, 2, & 3.
  cvec1.translate_ip(del);              // cvec1 itself translated by increments in pt del.
  }
```

See Also: trans_x, trans_y, trans_z

3.5.2 trans_x
3.5.3 trans_y
3.5.4 trans_z

Usage:

```
#include <Level1/coord_vec.h>
coord_vec trans_x(double delx)
void trans_x_ip(double delx)
coord_vec trans_y(double dely)
void trans_y_ip(double dely)
coord_vec trans_z(double delz)
void trans_z_ip(double delz)
```

Description:

The function ***trans_u*** allows the user to translate the entire vector along the Cartesian ***u***-axis where ***u*** ∈ ***{x,y,z}***.

$$pt(i)_{translated} = \begin{Bmatrix} x(i)_{translated} \\ y(i)_{translated} \\ z(i)_{translated} \end{Bmatrix} = \begin{Bmatrix} x(i) + delx \\ y(i) + dely \\ z(i) + delz \end{Bmatrix} = pt(i) + \begin{Bmatrix} delx \\ dely_i \\ delz_i \end{Bmatrix}$$

When ***u=x***, ***dely=delz=0***; when ***u=y***, ***delx=delz=0***, and when ***u=z***, ***delx=dely=0***. This makes sense as a translation only if the coordinates are Cartesian. The function leaves the input coordinate vector unaltered and translates a copy. However the functions having suffice ***_ip*** are "in-place", performing the translation directly on the coordinate vector, returning void.

Return Value:

Either a coordinate vector is returned or the function returns nothing.

Examples:

```
#include <gamma.h>
main()
  {
  coord_vec cvec, cvec1(1000);          // Define two coordinate vectors cvec and cvec1.
  cvec = cvec1.trans_x(-4);             // cvec set equal to cvec1 translated along x by -4.
  cvec1.trans_x_ip(1.30);               // cvec1 itself is translated along x by 1.3.
  cvec = cvec1.trans_y(-4);             // cvec set equal to cvec1 translated along y by -4.
  cvec1.trans_y_ip(1.30);               // cvec1 itself is translated along y by 1.3.
  cvec = cvec1.trans_z(-4);             // cvec set equal to cvec1 translated along z by -4.
  cvec1.trans_z_ip(1.30);               // cvec1 itself is translated along z by 1.3.
  }
```

See Also: translate

3.6 Projection Functions

3.6.1 project

Usage:

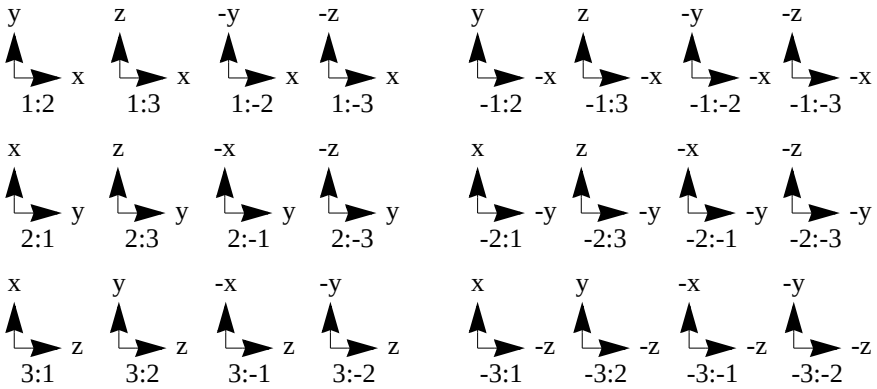
```
#include <Level1/coord_vec.h>
row_vector project(int projx, int projy)
```

Description:

The function ***project*** takes a vector of 3D-coordinates and returns a vector of 2D-coordinates. No complex mathematics is done to the coordinate values, one axis is simply thrown away. The values of ***projx*** and ***projy*** dictate which of the 3D axes are projected onto which of the 2D axes. The axes are designated as follows:

projx, projy:	1	-1	2	-2	3	-3
3D-axis:	x	-x	y	-y	z	-z

Thus, to project the 3D z-axis onto the 2D x-axis and the 3D y-axis onto the 2D negative y-axis the function would be called with arguments (3, -2). For clarity, the 2-axis from 3D space are shown in the next figure.



Above, the intergers are values for ***projx:projy***. The two axis labels, {x,y,z} are the 3D axis labels which are overlaid on the conventional x & y axes in 2D. The first shown, 1:2, is the common 3D -> 2D projection.

Return Value:

The function returns a row vector equal in size to the input coordinate vector.

Example:

```
#include <gamma.h>
main()
  {
  coord_vec cvec(1000);          // Define a coordinate vector cvec.
  row_vector vx;                 // Define a row vector vx.
  vx = cvec.project(-3,1);       // Project 3D negative z-axis to 2D x-axis (vx reals)
  }                              // and the 3D x-axis to the 2D y-axis (vx imags).
```

See Also:

3.7 Scalar Functions

3.7.1 *

3.7.2 *=

Usage:

```
#include <Level1/coord_vec.h>
friend coord_vec operator * (const coord_vec &cv, double r)
friend coord_vec operator *= (double r, const coord &cv )
friend void operator *= (coord_vec &cv, double r )
```

Description:

These functions multiply all ordinates of each point ***pt*** by the constant ***r***: ***{x,y,z} -> {rx, ry, rz}.***

Return Value: coordinate or void

Example:

```
#include <gamma.h>
main()
  {
  }
```

See Also: trans_x, trans_y, trans_z

3.7.3 /

3.7.4 /=

Usage:

```
#include <Level1/coord_vec.h>
friend coord_vec operator / (const coord_vec &cv, double r)
friend void operator /= (coord_vec &cv, double r )
```

Description:

These functions divide all ordinates of each point ***pt*** by the constant ***r***: ***{x,y,z} -> {x/r,y/r, z/r}.***

Return Value: coordinate or void

Example(s):

```
#include <gamma.h>
main()
  {
  }
```

See Also: trans_x, trans_y, trans_z

3.8 Auxiliary Functions

3.8.1 size

Usage:

```
#include <Level1/coord_vec.h>
int coord_vec::size ()
```

Description:

This function returns the size of the coordinate vector. The size of the vector is equivalent to the number of coordinates it contains.

Return Value:

An integer is returned.

Example(s):

```
#include <gamma.h>
main()
  {
  coord_vec cvec(100);                  // Define a coordinate vector cvec.
  cvec.read("test.cvec");               // Input vector coordinates from file test.cvec.
  cout << "\n# points: "<< cvec.size(); // Output the number of coordinates read in.
  }
```

See Also:

3.8.2 max_x

3.8.3 max_y

3.8.4 max_z

Usage:

```
#include <Level1/coord_vec.h>
double coord_vec::max_x ()
double coord_vec::max_y ()
double coord_vec::max_z ()
```

Description:

This function ***max_u*** returns the maximum value for component ***u*** over of all points in the vector. There are three distinct functions as $\boldsymbol{u} \in \{x,y,z\}$, ***x*** is the first ordinate, ***y*** the second, and ***z*** the third ordinate of the points.

Return Value:

A double is returned.

Example:

```
#include <gamma.h>
main()
  {
  coord_vec cvec;                       // define a coordinate vector cvec.
  cvec.read("test.cvec");               // input vector coordinates from file test.cvec.
```

```
cout << "\max x: "<< cvec.max_x();        // output the maximum x value of all points.
cout << "\max y: "<< cvec.max_y();    // output the maximum y value of all points.
cout << "\max z: "<< cvec.max_z();    // output the maximum z value of all points.
}
```

See Also: maxima

3.8.5 maxima

Usage:

```
#include <Level1/coord_vec.h>
coord coord_vec::maxima() const
void coord_vec::maxima(double &x, double &y, double& z) const
```

Description:

Function ***maxima*** returns the maximum ***x***, ***y***, and ***z*** values of all points in the coordinate vector. These values can either be returned in a single coordinate point or three double numbers can be input and will be set to the corresponding values by the function.

Return Value:

Either a coordinate is returned or nothing depending upon the argument list used.

Examples:

```
#include <gamma.h>
main()
  {
  coord_vec cvec(100);                  // Define a coordinate vector cvec.
  cvec.read("test.cvec");               // Input vector coordinates from file test.cvec.
  coord pt;                             // Make an empty coordinate
  pt = cvec.maxima();                   // pt contains maximum x, y, and z values.
  double x,y,z;                         // Make an empty coordinate
  cvec.maxima(x,y,z);                   // x, y and z set to maximum x, y, and z values.
  }
```

See Also: max_x, max_y, max_z

3.8.6 max_R

Usage:

```
#include <Level1/coord_vec.h>
double coord_vec::max_R ()
```

Description:

This function returns the maximum radius value over all points in the coordinate vector.

Return Value:

A double is returned.

Example:

```
#include <gamma.h>
main()
```

```
{
coord_vec cvec(100);                    // define a coordinate vector cvec.
cvec.read("test.cvec");                 // input vector coordinates from file test.cvec.
cout << "\max R: "<< cvec.max_R();  // output the maximum radius value of all points.
}
```

See Also: max_x, max_y, max_z

3.8.7 vectors

Usage:

```
#include <Level1/coord_vec.h>
coord_vec coord_vec::vectors ()
```

Description:

This function returns a coordinate vector containing the vectors of the original coordinate vector. Mathematically, each element of the returned matrix ***mx*** is given by

$$\langle i|mx|j\rangle = d_{ij},$$

where the distance between the points ***i*** and ***j***, d_{ij} is given by

$$d_{ij} = \sqrt{x_{ij}^2 + y_{ij}^2 + z_{ij}^2} \qquad z_{ij} = z_j - z_i.$$

Return Value:

A coordinate vector is returned.

Example(s):

```
#include <gamma.h>
main()
  {
  coord_vec coords;                     // declare a coordinate vector named coords
  coords.read("coord.file");            // read coordinate vector from file named coord.file
  cout << "\nTheta matrix (Radians)\n";
  cout << coords.thetas() << "\n";      // output the theta values in radians
  cout << "\nTheta matrix (degrees)\n";
  cout << coords.thetas(1) << "\n";     // output the theta values in degrees
  }
```

See Also: thetas, phis

3.8.8 distances

Usage:

```
#include <Level1/coord_vec.h>
matrix coord_vec::distances (int Angs=0)
```

Description:

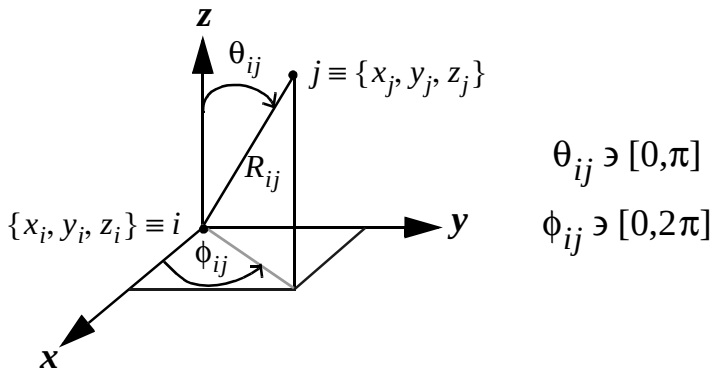
This function returns a matrix containing the values of the lengths of the vectors connecting each of the points in the coord_vec. Mathematically, each element of the returned matrix ***mx*** is given by

$$<i|mx|j> = d_{ij},$$

where the distance between the points ***i*** and ***j***, d_{ij} is given by

$$d_{ij} = \sqrt{x_{ij}^2 + y_{ij}^2 + z_{ij}^2} \qquad z_{ij} = z_j - z.$$

The flag ***Angs*** can be optionally used as a function argument. When set to a non-zero value the matrix elements are output in Angstroms rather than the default units of meters[1]. The returned matrix will be square and have a dimension equivalent to the number of coordinates in the coordinate vector. The standard coordinate system used in GAMMA is given in Figure 3-A on page 57, reproduced below to detail the function.



Return Value:

A matrix is returned.

Example(s):

```
#include <gamma.h>
main()
  {
  coord_vec coords;                    // declare a coordinate vector named coords
  coords.read("coord.file");           // read coordinate vector from file named coord.file
  cout << "\nTheta matrix (Radians)\n";
  cout << coords.thetas() << "\n";     // output the theta values in radians
  cout << "\nTheta matrix (degrees)\n";
  cout << coords.thetas(1) << "\n";    // output the theta values in degrees
  }
```

See Also: thetas, phis

3.8.9 thetas

Usage:

```
#include <Level1/coord_vec.h>
matrix coord_vec::thetas (int deg=0)
```

1. This assumes that each coordinate is stored into the coordinate vector originally in meters. If not, setting the ***Angs*** flag scales the output distances by a factor of 10^{10}. It is wise to maintain all coordinates in SI units, for Cartesian coordinates this is meters. Provisions are normally made for using more common units in all functions performing I/O.

Description:

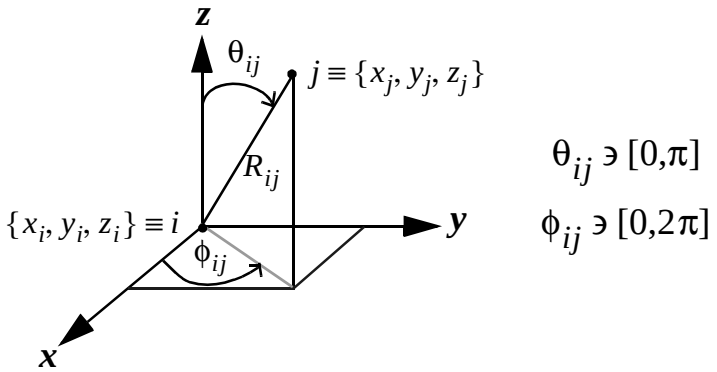
This function returns a matrix containing the values of the spherical theta angles for the vectors connecting each of the points in the ***coord_vec***. Mathematically, each element of the returned matrix ***mx*** is given by

$$\theta_{ij} = \langle i|mx|j\rangle = \mathrm{acos}(z_{ij}/R_{ij}),$$

where the distance between the points ***i*** and ***j***, R_{ij} is given by

$$R_{ij} = \sqrt{x_{ij}^2 + y_{ij}^2 + z_{ij}^2} \qquad z_{ij} = z_j - z_i .$$

The flag ***deg*** can be optionally used as a function argument. When set to a non-zero value the matrix elements are output in degrees rather than the default units of radians. The returned matrix will be square and have a dimension equivalent to the number of coordinates in the coordinate vector. The standard coordinate system used in GAMMA is given in Figure 3-A on page 57, reproduced below to detail the function.



Return Value:

A matrix is returned.

Examples:

```
#include <gamma.h>
main()
  {
  coord_vec coords;                    // declare a coordinate vector named coords
  coords.read("coord.file");           // read coordinate vector from file named coord.file
  cout << "\nTheta matrix (Radians)\n";
  cout << coords.thetas() << "\n";     // output the theta values in radians
  cout << "\nTheta matrix (degrees)\n";
  cout << coords.thetas(1) << "\n";    // output the theta values in degrees
  }
```

See Also: distances, phis

3.8.10 phis

Usage:

```
#include <Level1/coord_vec.h>
matrix coord_vec::phis (int deg=0)
```

Description:

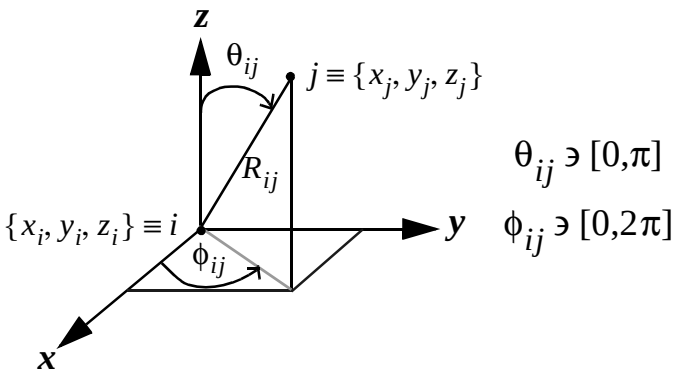
This function returns a matrix containing the values of the spherical phi angles for the vectors connecting each of the points in the coord_vec. Mathematically, each element of the returned matrix ***mx*** is given by

$$\phi_{ij} = \langle i|mx|j\rangle = \text{atan}(y_{ij}/x_{ij}) ,$$

where the between the points $\boldsymbol{i}$ and $\boldsymbol{j}$, $\boldsymbol{y_{ij}}$ is given by

$$\boldsymbol{y_{ij} = y_j - y_i}.$$

The flag ***deg*** can be optionally used as a function argument. When set to a non-zero value the matrix elements are output in degrees rather than the default units of radians. The returned matrix will be square and have a dimension equivalent to the number of coordinates in the coordinate vector. The standard coordinate system used in GAMMA is given in Figure 3-A on page 57, reproduced below to detail the function.



Return Value:

A matrix is returned.

Example(s):

```
#include <gamma.h>
main()
  {
  coord_vec coords;                    // declare a coordinate vector named coords
  coords.read("coord.file");           // read coordinate vector from file named coord.file
  cout << "\nPhi matrix (Radians)\n";
  cout << coords.phis() << "\n";       // output the phi values in radians
  cout << "\nPhi matrix (degrees)\n";
  cout << coords.phis(1) << "\n";      // output the phi values in degrees
  }
```

See Also: distances, thetas

3.9 Coordinate Access

3.9.1 ()

Usage:

#include <Level1/coord_vec.h>
coord coord_vec::operator() (int index)

Description:

This function returns the coordinate at the index specified.Return Value:

Return Value:

A coordinate is returned.

Example:

```
#include <gamma.h>
main()
  {
  coord_vec cvec(20);                 // Make an empty coord_vec with 20 coordinates.
  cout << cvec(19);                   // Output the last coordinate, remember [0,19] index.
  }
```

See Also: put, get, x, y, z

3.9.2 put

Usage:

#include <Level1/coord_vec.h>
void coord_vec::put(const coord& pt, int index)
void coord_vec::put(double x, double y, double z, int index)

Description:

This function sets the coordinate at the index specified by ***index*** to the value given by either a coordinate point ***pt*** or by three values ***x***, ***y***, and ***z***.

Return Value:

None, the function is void.

Examples:

```
#include <gamma.h>
main()
  {
  coord_vec cvec(20);                 // Make an empty coord_vec with 20 coordinates.
  coord pt(1,1,1);                    // Make a coordinate with values x=y=z=1.
  cvec.put(pt, 10);                   // Set the 11th point in cvec to the values in pt.
  coord pt(2,-1,7, 11);               // Set the 12th point in cvec to x=2, y=-1, z=7.
  }
```

See Also: (), get, x, y, z

3.9.3 get

Usage:

```
#include <Level1/coord_vec.h>
coord coord_vec::get(int index)
```

Description:

This function retrieves the coordinate at the index specified by ***index***.

Return Value:

The function returns a coordinate.

Example:

```
#include <gamma.h>
main()
  {
  coord_vec cvec(20);          // Make an empty coord_vec with 20 coordinates.
  cout << cvec.get(19);        // Output the last coordinate, remember [0,19] index.
  }
```

See Also: (), put, x, y, z

3.9.4 x

3.9.5 y

3.9.6 z

Usage:

```
#include <Level1/coord_vec.h>
double coord_vec::x (int index) const
void coord_vec::x (int index, double xin)
double coord_vec::y (int index) const
void coord_vec::y (int index, double yin)
double coord_vec::z (int index) const
void coord_vec::z (int index, double zin)
```

Description:

This function ***u*** either returns or sets the ***u***-component value of the coordinate point specified by ***index***. Here $u \in \{x,y,z\}$ and ***x*** is the first ordinate, ***y*** the second, and ***z*** the third ordinate.

Return Value:

Depending upon the argument list the function is void or returns a double.

Examples:

```
#include <gamma.h>
main()
  {
  coord_vec cvec(10);          // Produce an empty coordinate vector with 10 points.
  cvec.x(0,-3.7);              // Set the x value of the first point to -3.7.
  cout << cvec.x(0);           // Output the x component of the first point.
  cvec.y(50,-13.7);            // Set the y value of the 51st t point to -13.7.
```

```
cout << cvec.y(0);          // Output the y component of the first point.
cvec.y(88,3.72);            // Set the z value of the 89th point to 3.72.
cout << cvec.z(0);          // Output the z component of the first point.
}
```

See Also: (), get, put

3.9.7 Rad

Usage:

```
#include <Level1/coord_vec.h>
coord::Rad ()
```

Description:

Function ***Rad*** either returns or sets the value of the radius of a coordinate point.

1. coord.z() - returns the value of the z component of the coordinate point.
2. coord.z(double zin) - sets the value of the z component of the coordinate point.

This function returns the value of the radius spherical component of a coordinate point.

Return Value:

A double precision number is returned.

Example(s):

```
#include <gamma.h>
main()
  {
  coord pt(-11,0,7);        // produces a coordinate pt at x=-11, y=0, z=7.
  cout << pt.Rad();         // radius sent to standard output sqrt(11**2 + 7**2).
  }
```

See Also: theta, phi

3.10 Parameter Functions

3.10.1 ()

Usage:

#include <Level1/coord_vec.h>
operator coord_vec::ParameterSet() const

Description:

This allows for the conversion of a coordinate vector into a GAMMA parameter set.

Return Value:

void. Used for conversions.

Example(s):

```
#include <gamma.h>
main()
  {
  }
```

See Also:

3.10.2 +=

Usage:

#include <Level1/coord_vec.h>
friend void coord_vec::operator+= (ParameterSet& pset, const coord_vec& cvec)

Description:

This allows for the addition of a coordinate vector to an existing parameter set.

Return Value:

void.

Example(s):

```
#include <gamma.h>
main()
  {
  }
```

See Also:

3.10.3 =

Usage:

#include <Level1/coord_vec.h>
int coord_vec::operator = (const ParameterSet& pset)

Description:

This allows for the assignment of a parameter set to a coordinate vector .

Return Value:

A modified parameter set.

Example(s):

```
#include <gamma.h>
main()
  {
  }
```

See Also:

3.11 Output Functions

3.11.1 print

Usage:

```
#include <Level1/coord_vec.h>
ostream& coord_vec::print(ostream &out) const
```

Description:

Outputs the coordinate vector to the output stream specified by ***out***. An optional argument ***units*** can be specified which will automatically scale the output for commonly used unit types

units	scale factor	assumed
0	1.0	none/meters
1	10^{10}	Angstroms
2	10^{9}	nanometers

Thus, ***units*** is useful when GAMMA performs computations with coordinate values in one type of unit (namely SI units - meters) but the user performs output on a different scale (e.g. Angstroms).[1]

Return Value:

None, the function is void.

Example:

```
#include <gamma.h>
main()
  {
  coord_vec cvec(100);          // define a coordinate vector cvec.
  cvec.print(cout);             // output all vector coordinates to cout.
  }
```

See Also: read, write, <<

3.11.2 <<

Usage:

```
#include <Level1/coord_vec.h>
ostream& operator << (ostream& ostr, coord_vec &cvec)
```

1. When coordinates are read in from an external file no specific units are assumed. On the other hand many functions in class ***coord_vec*** assume the input values are relative to a Cartesian coordinate system. Furthermore, classes derived from ***coord_vec*** may indeed assume specific units on input. When this occurs it will hopefully be document in the description of the functions used.

Description:

Outputs the coordinate vector to standard output.

Return Value:

None, the function is void.

Example:

```
#include <gamma.h>
main()
  {
  coord_vec cvec(100);                  // define a coordinate vector cvec.
  cout << "\nCoordinates:\n" << cvec;    // output all vector coordinates to cout.
  }
```

See Also: read, write, print

3.11.3 write

Usage:

#include <Level1/coord_vec.h>

void coord_vec::write(const string &filename) const

Description:

Outputs the coordinate vector to a file with the filename given. The coordinate vector is output in the GAMMA parameter set format. The coordinate vector can be re-read from the file with the function read. Note that this is an inefficient way to perform disk storage of large coordinate vectors. Also, the file opened and closed with the function call. Any previously existing file with the same name will be overwritten.

Return Value:

None, the function is void.

Example:

```
#include <gamma.h>
main()
  {
  coord_vec cvec(100);                  // define a coordinate vector cvec.
  cvec.write("test.cvec");              // output vector coordinates to file named test.cvec.
  }
```

See Also: read, print, <<

3.12 Input Functions

3.12.1 read

Usage:

```
#include <Level1/coord_vec.h>
int coord_vec::read(const string& filename, int idx=-1, int warn=2)
int coord_vec::read(ParameterSet & pset, int idx=-1, int warn=2)
```

Description:

Reads the coordinate vector from either a specified input file ***filename*** or from a parameter set ***pset***. The file must be a valid parameter set. The file or the parameter set must contain the parameters required to define one or more coordinate vectors. The optional argument ***idx*** allow the reading of a coordinate vector whose parameter names have the prefix [#] where #=***idx***.The default, ***idx***=-1, will use NO such prefix. The last (optional) argument, ***warn***, allows the user to set the warning level used when the vector cannot be read from the input file or parameter set. The default, ***warn***=2, will cause a fatal error if the vector cannot be read. The value of 1 will output error messages but not cause the program to stop. The value of 0 cause no error messages nor program stoppage. The function will return 1 if the read was successful or 0 if unsuccessful.

Return Value:

TRUE/FALSE.

Example:

```
#include <gamma.h>
main()
  {
  coord_vec cvec;                    // Define a coordinate vector cvec
  cvec.read("test.pset");            // Read coordinates from file test.pset
  cvec.read("test.pset", 2);         // Read coordinates with [2] prefix from file test.pset
  ParameterSet pset;                 // Define a parameter set
  pset.read("test.pset");            // Read all parameters if file test.pset
  cvec.read(pset, 0);                // Read coordinates with [0] prefix from parameter set
  }
```

See Also: write, print, <<, ask_read

3.12.2 ask_read

Usage:

```
#include <Level1/coord_vec.h>
string coord_vec::ask_read(int argc, char* argv[], int argn, int idx=-1, int warn=2)
```

Description:

Interactively ask for or reads the coordinate vector from a specified input file. The file must be a valid parameter set. The value ***argc*** should be the number of commands supplied when the program is executed and the array ***argv*** should contain those commands. The value of ***argn*** will be the argument containing the name of the parameter file. If no argument was supplied the filename is requested from the user. The optional argument ***idx*** allow the reading of a coordinate vector whose parameter names have the prefix [#] where #=***idx***.The default, ***idx***=-1, will use NO such prefix. The last (optional) argument, ***warn***, allows the user to set the warning level used when the vector can-

not be read from the input file. The default, ***warn***=2, will cause a fatal error if the vector cannot be read. The value of 1 will output error messages but not cause the program to stop. The value of 0 cause no error messages nor program stoppage. The function will return the filename it (attempted to) read.

Return Value:

TRUE/FALSE.

Example:

```
#include <gamma.h>
main(int argc, char* argv[])
  {
  coord_vec cvec;                    // Define a coordinate vector cvec
  int qn =1;                         // Input parameter index
  cvec.ask_read(argc,argv,qn++);     // Ask for/Read coordinates from file (cmd 1)
  cout << cvec << "\n\n"             // Have a look at the vector
  }
```

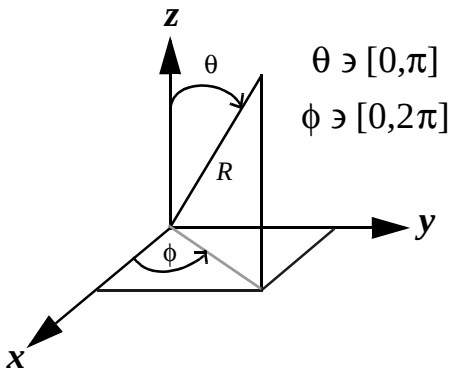
See Also: write, print, <<, read

3.13 Description

Class ***Coordinate Vector*** facilitates the manipulation of a list of coordinate points (See class ***coord***). Each coordinate contains three values: x,y, and z in Cartesian space, for example. Although the coordinates need not be Cartesian, specific functions must implicitly know the type of coordinates they are dealing with. There are functions are provided to rotate, translate, and perform a variety of common coordinate transformations simultaneously over all coordinates in the vector and these mu. See class *coord* for information concerning specifics on individual coordinates.

The most common coordinate system is the one shown below, the right-handed Cartesian system. Furthermore, spherical angles are overlaid on the Cartesian system and their ranges given.[1]

Cartesian and Spherical Coordinate Systems



$$R = \sqrt{x^2 + y^2 + z^2} \qquad x = R\sin\theta\cos\phi$$
$$\theta = \operatorname{acos}(z/R) \qquad y = R\sin\theta\sin\phi$$
$$\phi = \operatorname{atan}(y/x) \qquad z = R\cos\theta$$

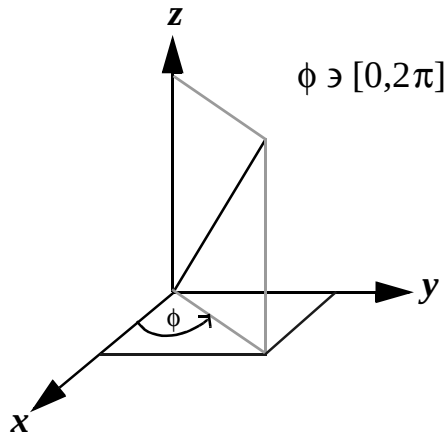
Figure 3-A : The right handed Cartesian axes are shown as well as the spherical angles and radius.

This class uses these systems in functions which implicitly assume (or use) either Cartesian or spherical coordinates. Of course, because the inverse trigonometric functions always return their principle values, the equations for θ and ϕ and cannot be used blindly.

Two other systems are the cylindrical coordinate system and the polar space coordinate system. These are shown in the following figure along with the appropriate relationships to the standard Cartesian system.

1. Many texts switch the angle designations of θ and ϕ from that shown in the figure. The usage here is consistent with angle designations in other chapters.

Cartesian and Cylindrical Coordinate Systems

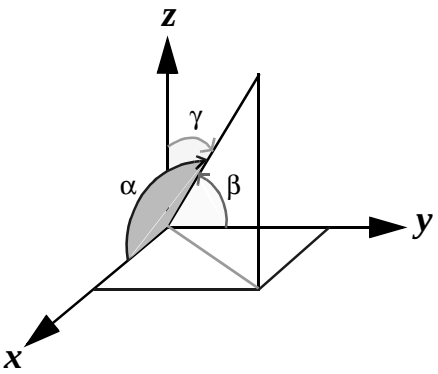


$$R = \sqrt{x^2 + y^2} \qquad x = R\cos\phi$$
$$z = z \qquad y = R\sin\phi$$
$$\phi = \operatorname{atan}(y/x) \qquad z = z$$

Figure 3-B

This class uses these systems in functions which implicitly assume (or use) either Cartesian or cylindrical coordinates.

Cartesian and Polar Space Coordinate Systems



$$R = \sqrt{x^2 + y^2 + z^2} \qquad x = R\cos\alpha$$
$$\alpha = \operatorname{acos}(x/R) \qquad y = R\cos\beta$$
$$\beta = \operatorname{acos}(y/R) \qquad z = R\cos\gamma$$
$$\gamma = \operatorname{acos}(z/R)$$

Figure 3-C

This class uses these systems in functions which implicitly assume (or use) either Cartesian or spherical coordinates.

3.14 Parameters

This section describes how an ASCII file may be constructed that is self-readable by a coordinate vector. The file can be created with any editor and is read with the coordinate vector member function “read” (or ask_read). The particular formats of individual coordinates are designated in the class ***coord***[1]. There may be additional alterations.

Number of Coordinates

The number of coordinates in a coordinate vector can be specified with the parameter NCoords. If this integer is not specified, the vector will attempt to determine the number of coordinates in the parameter set. It will begin with Coord(0) and step through Coord(1), Coord(2), and so on until the first coordinate is missing. If more coordinates are specified than the number set, they will be ignored.

Number Of Coordinates Parameters

Parameter	Assumed Units	Example ASCII Lines Parameter (Type=3) : Value - Statement
NCoords	none	NCoords (0) : 5 - Number of Coordinates
NSpins	none	NSpin (0) : 3 - Number of Spins (Spin Coordinates)

Figure 3-D Parameters for the number of coordinates in a coordinate vector. Type 0 indicates an integer. The parameter NSpins is provided with compatibility in MR simulations using spin systems. NCoords will be preferentially used if both NCoords & NSpins are set.

Cartesian & Arbitrary Coordinates

Cartesian coordinates can be input either in Angstroms (default), nanometers, or meters depending upon the parameter name. GAMMA will internally scale the former two and store all in meters. Arbitrary values should be input in ***meters*** as these will NOT be scaled so that the user may apply his/her own scaling if necessary.

Cartesian Coordinate Parameters

Parameter	Assumed Units	Example ASCII Lines Parameter (Type=3) : Value - Statement
Coord	A	Coord(0) (3) : (0.0, 0.0, 1.0) - Coordinate in Angstroms
Coordn	nm	Coordn(1) (3) : (2.0, 1.0, 1.0) - Coordinate in nanometers
Coordm	none, m	Coordm(2) (3) : (0.0, 0.0, 1.0) - Coordinate in meters

Figure 3-E Parameters for coordinates in a Cartesian mode. For general (unscaled) coordinates use Coordm. Type 3 indicates a coordinate parameter.

1. Coordinate formats are part of the class coord specification. Alterations in class coord will automatically apply to coordinate vectors. See class coord for the most recent documentation.

Spherical Coordinates

Spherical coordinates can be input with the radius either in Angstroms (default), nanometers, or meters and with the angles in either degrees (default) or radians. Again, how the input values are interpreted depends upon the parameter name. GAMMA will internally store all of these coordinates in a Cartesian axis system with meter units

Spherical Coordinate Parameters

Parameter	Assumed Units	Example ASCII Lines Parameter (Type=3) : Value - Statement
CoordSph	A, deg	CoordSph(0) (3) : (1.0, 0.0, 0.0) - R is 1 Angstrom
CoordSphn	nm, deg	CoordSphn(1) (3) : (2.0, 90.0, 0.0) - R = 2 nm, θ=90
CoordSphm	m, deg	CoordSphm(2) (3) : (0.1, 0.0, 90.0) - R = 1m, ϕ=90
CoordSphR	A, rad	CoordSphR(0) (3) : (1.0, 0.0, 0.0) - R is 1 Angstrom
CoordSphRn	nm, rad	CoordSphRn(1) (3) : (2.0, 1.57, 0.0) - R = 2 nm, θ=180
CoordSphm	m, rad	CoordSphRm(2) (3) : (0.1, 0.0, 0.79) - R = 1m, ϕ=90

Figure 3-F Shown are three possible parameters used to set a coordinates in a Spherical mode.

Cylindrical Coordinates

Cylindrical coordinates can be input with the height & radius either in Angstroms (default), nanometers, or meters and with the angle in either degrees (default) or radians. How the input values are interpreted depends upon the parameter name. GAMMA will internally store all of these coordinates in a Cartesian axis system with meter units

Cylindrical Coordinate Parameters

Parameter	Assumed Units	Example ASCII Lines Parameter (Type=3) : Value - Statement
CoordCyl	A, deg	CoordCyl(0) (3) : (1.0, 0.0, 0.0) - R is 1 Angstrom
CoordCyln	nm, deg	CoordCyln(1) (3) : (2.0, 90.0, 0.0) - R = 2 nm, θ=90
CoordCylm	m, deg	CoordCylm(2) (3) : (0.1, 0.0, 9.0) - R = 1m, z=9m
CoordCylR	A, rad	CoordCylR(0) (3) : (1.0, 0.0, 0.0) - R is 1 Angstrom
CoordCylRn	nm, rad	CoordCylRn(1) (3) : (2.0, 1.57, 0.0) - R = 2 nm, θ=90
CoordCylm	m, rad	CoordCylRm(2) (3) : (0.1, 0.0, 0.79) - R = 1m, z=0.79m

Figure 3-G Shown are three possible parameters used to set a coordinates in a Cylindrical mode.

The following input file and program indicate how users may read coordinates from such files.

Simple Read of Coordinates From An ASCII File

CoordI0.pset

```
This is the default input file for the program CoordI0.cc. It demonstrates how to set up
individual 3D coordinates. The most important aspect to note is that the RADIANS
units should be input with good precision!

NCoords     (0): 13 - Number of coordinates

                    Cartesian Coordinates
Coord(0)      (3) : (0.0, 0.0, 2.0)   - Coordinate (Angstroms)
Coordn(1)     (3) : (0.0, 2.0, 0.0)   - Coordinate (nanometers)
Coordm(2)     (3) : (2.0, 0.0, 0.0)   - Coordinate (meters/none)
                    Spherical Coordinates
CoordSph(3)      (3) : (2.0, 0.0,  90.0) - Coordinate [r(A),t(deg),f(deg)]
CoordSphn(4)     (3) : (2.0, 90.0, 90.0) - Coordinate [r(nm), t(deg), f(deg)]
CoordSphm(5)     (3) : (2.0, 90.0, 0.0)  - Coordinate [r(m), t(deg), f(deg)]
CoordSphR(6)     (3) : (2.0, 0.0,        1.570796327)  - Coordinate [r(A),t(rad),z(A)]
CoordSphRn(7)    (3) : (2.0, 1.570796327, 1.570796327)  - Coordinate [r(nm), t(rad), f(rad)]
CoordSphRm(8)    (3) : (2.0, 1.570796327, 0.0)       - Coordinate [r(m), t(rad), f(rad)]
                    Cylindrical Coordinates
CoordCyl(9)      (3) : (0.0, 0.0,  2.0)  - Coordinate [r(A),t(deg),z(A)]
CoordCyln(10)    (3) : (2.0, 90.0, 0.0)  - Coordinate [r(nm), t(deg), z(nm)]
CoordCylm(11)    (3) : (2.0, 0.0,  0.0)  - Coordinate [r(m), t(deg), z(nm)]
CoordCylR(12)    (3) : (0.0, 0.0, 2.0)         - Coordinate [r(A),t(rad),f(rad)]
```

CoordI0.cc

```
#include <gamma.h>

main()
 {
 string filename = "CoordI0.pset";
 coord_vec cvec;
 cvec.read(filename)
 cout << cvec;
 cout << "\n\n";
 }
```

Figure 3-H Specifying coordinates using an external ASCII file. Note that when angles are input in radians it is important to specify their values with good precision.

Multiple coordinate vectors may be specified within the same ASCII file by using the prefix [#] on each of the coordinate parameter names in combination with that same # in the read statement. A simple example is shown in the next figure. The # -1 is reserved for implying no prefix.

Multiple Coordinate Vector Read From An ASCII File

CoordI2.pset

```
This is the default input file for the program CoordI2.cc. It demonstrates
 how to set up multiple coordinate vectors in the same file.


NCoords      (0) : 3                  - Number of coordinates
Coord(0)      (3) : (0.0, 0.0, 2.0)   - Coordinate (Angstroms)
Coord(1)      (3) : (0.0, 2.0, 0.0)   - Coordinate (Angstroms)
Coord(2)      (3) : (2.0, 0.0, 0.0)   - Coordinate (Angstroms)
[0]CoordSph(0)   (3) : (2.0, 0.0,  90.0)  - Coordinate [r(A),θ(deg),ϕ(deg)]
[0]CoordSphn(1)  (3) : (2.0, 90.0, 90.0) - Coordinate [r(nm), θ(deg), ϕ(deg)]
[0]CoordSphm(2) (3) : (2.0, 90.0, 0.0)   - Coordinate [r(m), θ(deg), ϕ(deg)]
[2]NCoords           (0) : 3                  - Number of coordinates
[2]NoordCyl(0)     (3) : (0.0, 0.0,  2.0)   - Coordinate [r(A),θ(deg),z(A)]
[2]NoordCyln(1)   (3) : (2.0, 90.0, 0.0)  - Coordinate [r(nm), θ(deg), z(nm)]
[2]NoordCylm(2)   (3) : (2.0, 0.0,  0.0)   - Coordinate [r(m), θ(deg), z(nm)]
```

CoordI2.cc

```
#include <gamma.h>

main()
 {
 string filename = "CoordI2.pset";
 coord_vec cvec;
 cvec.read(filename)
 coord_vec cvec0;
 cvec.read(filename, 0)
 coord_vec cvec2;
 cvec.read(filename, 2)
 }
```

Figure 3-I Specifying multiple coordinates using an external ASCII file. Note that the index used in the [#] parameter name prefixes must match that of the index in the read function.

4 Exponential

4.1 Overview

The ***Exponential*** module contains functions pertaining to the use of exponential functions in GAMMA.

4.2 Available Functions

4.3 Description Sections

4.4 Exponential Figures

4.5 Exponential Functions

4.5.1 Exponential

Usage:

#include <Exponential.h>

row_vector Exponential(int npts, double W, double R)

row_vector Exponential(int npts, double time, double W, double RT, int type=0)

Description:

The function ***Exponential*** is used to create a 1-dimensional array containing a discrete exponential function.

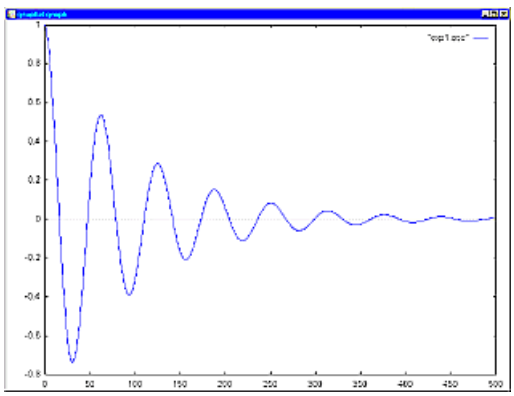
1. Exponential(int npts, double W, double R) - Creates a vector of ***npts*** points filled with an exponential which oscillates at the rate ***W*** and decays at the rate of ***R***. Both ***W*** and ***R*** are assumed in units of ***1/pt.***
2. Exponential(int npts, double time, double W, double RT, int type=0) - Creates a vector of ***npts*** points filled with an exponential spanning the time [0, ***time***) which oscillates at the rate ***W*** and decays at the rate related to ***RT***. When the value of ***type*** is zero (default) ***RT*** is assumed a decay rate. When ***type*** is nonzero, ***RT*** is assumed to be a time. ***W*** is taken in units of ***1/(time units)*** whereas ***RT*** will either be the same as ***W*** or the inverse (same as ***time*** unts) depending upon ***type.***

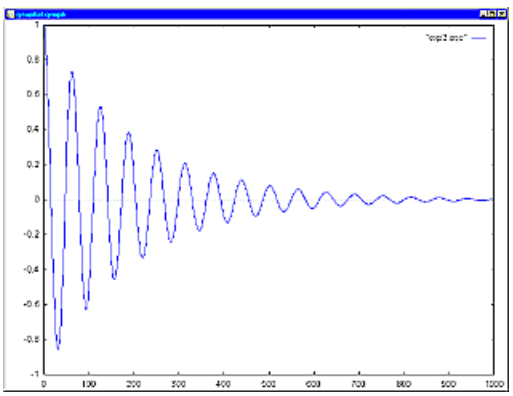
Return:

row_vector

Examples:

```
#include <gamma.h>
main()
  {
  row_vector vx;                          // A row vector
  vx = Exponential(500,0.1,0.01);         // 500 point exponential, W=0.1/pt, R=0.01/pt
  GP_1D("exp1.asc", vx);                  // Output, ASCII for Gnuplot
  GP_1Dplot("exp1.gnu", "exp1.asc");      // Plot to screen
  vx = Exponential(1000,1.0,100.0,5.0);   // 1000 pts, length 1sec, W=100/(2*PI) Hz, R=5/sec
  GP_1D("exp2.asc", vx);                  // Output, ASCII for Gnuplot
   GP_1Dplot("exp2.gnu", "exp2.asc");     // Plot to screen
  }
```





See Also: DExponential

4.5.2 DExponential

Usage:

#include <Exponential.h>

row_vector DExponential(int npts, double W, double R)

row_vector DExponential(int npts, double time, double W, double RT, int type=0)

Description:

The function ***DExponential*** is used to create a 1-dimensional array containing a discrete derivative exponential function.

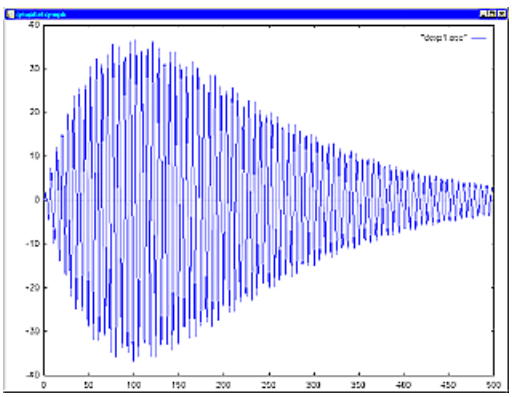
1. DExponential(int npts, double W, double R) - Creates an vector of ***npts*** points filled with a differential exponential oscillating at rate ***W*** and decaying at the rate of ***R***. Both ***W*** and ***R*** are assumed in units of ***1/pt.***
2. DExponential(int npts, double time, double W, double RT, int type=0) - Creates an vector of ***npts*** points filled with a differential exponential spanning the time [0, ***time***) which oscillates at the rate ***W*** and decays at the rate related to ***RT***. When the value of type is zero (default) ***RT*** is assumed a decay rate. When ***type*** is nonzero, ***RT*** is assumed to be a time. ***W*** is taken in units of ***1/(time units)*** whereas ***RT*** will either be the same as ***W*** or the inverse (same as ***time*** unts) depending upon ***type.***

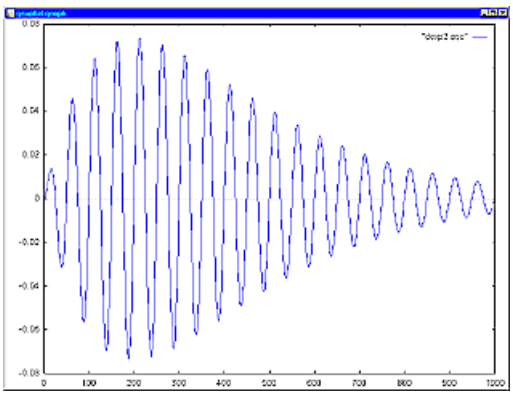
Return Value:

row_vector

Examples:

```
#include <gamma.h>
main()
  {
  row_vector vx;                          // A row vector
  vx = DExponential(500,1.0,0.01);        // 500 point exponential, W=0.1/pt, R=0.01/pt
  GP_1D("dexp1.asc", vx);                 // Output, ASCII for Gnuplot
  GP_1Dplot("dexp1.gnu", "dexp1.asc");    // Plot to screen
  vx = Exponential(1000,1.0,100.0,5.0);   // 1000 pts, length 1sec, W=100/(2*PI) Hz, R=5/sec
  GP_1D("exp2.asc", vx);                  // Output, ASCII for Gnuplot
   GP_1Dplot("exp2.gnu", "exp2.asc");     // Plot to screen
  }
```





See Also: Exponential

4.5.3 Exponen_cut

Usage:

#include <Exponential.h>

int Expon_cut(int npts, double time, double W, double R, double cutoff=1.e-10)

void Expon_cut(int* ihi, matrix& mx, double tinc, int npts, double cutoff=1.e-10)

Description:

The function ***Exponen_cut*** is used to determine at which point an exponential's magnitude falls below a specified cutoff. The cutoff is given by the argument ***cutoff*** which must span [0, 1]. The exponential(s) are ***npts*** points long.

1. Expon_cut(int npts, double time, double W, double R, double cutoff=1.e-10) - Returns the point where the exponential falls below ***cutoff***. The exponential which oscillates at the rate ***W*** and decays at the rate of ***R*** and spans the time ***time.***
2. Expon_cut(int* ihi, matrix& mx, double tinc, int npts, double cutoff=1.e-10) - Fills the integer vector ***ihi*** with cutoff point indices where magnitudes of the exponentials defined in array ***mx*** fall below ***cutoff***. The exponentials all have time ***tinc*** between points***.***

Return Value:

Void.

Examples:

See Also: Exponential, DExponential

4.6 Description

4.6.1 Overview

Exponentials are commonly found in magnetic resonance because they are related to driven oscillators. Damped oscillations are mathematically represented by decaying complex exponentials in the time domain. The ***Exponential*** module contains functions pertaining to the use of exponential functions in GAMMA.

4.6.2 Analog Exponential

The general form of a complex exponential function, as defined in GAMMA, is given by

$$Exp(t) = e^{i(\omega_o + iR)t} = e^{i(\omega_o + i/T_2)t} \tag{4-1}$$

Here, ω_o will be the oscillation frequency, R the decay rate, and T_2 the (relaxation) time constant. For mathematical purposes it is best to keep both ω_o and R in radians/sec (which avoids having factors of π in equations), however Hertz is generally preferred in MR work. We will assume the former herein and point out the conversions required (if any) to switch into other units. In addition time is taken as a non-negative number, i.e. one should not use the function at negative times.

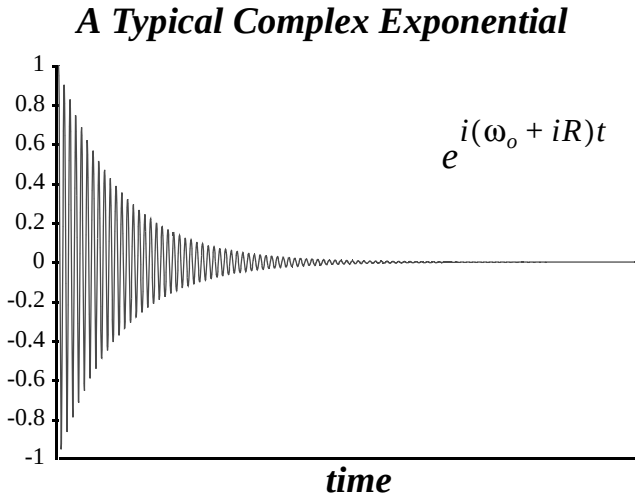


Figure A Depiction of a complex exponential. Only the reals are plotted[1].

Note how the above plot begins a time zero and extends to infinite time. Time is taken as a non-negative number, i.e. in real situations we do not have negative times. A second aspect is that we have used a non-negative value of ***R***. Again, in experimental situations our oscillations normally decay to zero in time. No decay would be the ***R=0*** case. A negative value for ***R*** would imply that the exponential goes to infinite amplitude as time goes to infinity. The frequencies can be any value. The maximum of the real part of a decaying exponential function is found at ***t=0*** and has amplitude of 1.

1. Plot created by the program Exponent0.cc on page 76

4.6.3 Exponential Decay Rates

A common way to characterize the rate ***R*** is to give its inverse time. In MR, ***R*** will be a relaxation rate and its inverse a relaxation time.

$$e^{i(\omega_o + iR)t} = e^{i(\omega_o + i/T_2)t} \tag{4-2}$$

When the time is equal to the relaxation time, $t=T_2$, the exponential magnitude has become ***1/e***. Using a non-oscillating (real) exponential, this value is easily seen from the function plot.

The Exponential Intensity a Time $T_2 = 1/R$

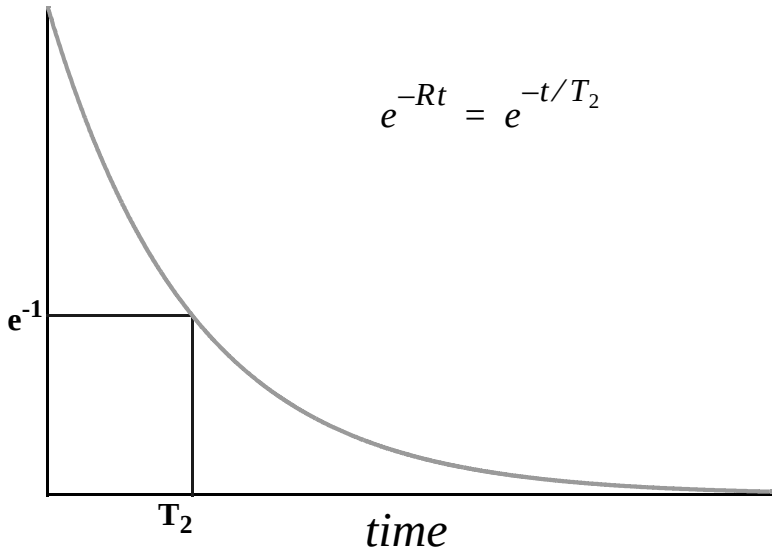


Figure B Depiction of real decaying exponential (ω_o=0) with T_2 time marked[1].

This is readily derived by evaluating the magnitude (or real part) of the function at the time T_2.

$$\|Exp(t)\| = \sqrt{e^{i(\omega_o + iR)t} \cdot e^{-i(\omega_o - iR)t}} = e^{-Rt}$$

$$\|Exp(t)\|_{t = T_2} = e^{-RT_2} = e^{-1} = \frac{1}{e} = 0.36788$$

4.6.4 Complex Lorentzian Relationship

A Fourier transformation on a complex exponential will produce a Lorentzian function. This is often performed because the frequency ω_o is easily seen in the frequency domain (the Lorentzian maximum) and the decay rate readily obtained (from the Lorentzian linewidth).

We will only sketch the mathematical details of the Fourier transform. An explicit derivation can be found in the GAMMA documentation of Lorentzian functions.

1. Plot created by the program Exponent0.cc on page 76

It Fourier transform is given by

$$FFT[Exp(t)] = \int_{-\infty}^{\infty} H(t) e^{i(\omega_o + iR)t} e^{-i\omega t} dt = \frac{1}{R + i(\omega - \omega_o)} = L(\omega) \qquad (4\text{-}3)$$

Since the exponential does not span the Fourier integral range of ±∞, use is made of the Heavyside function, ***H(t)*** which is zero until it reaches ***t=0*** and 1 thereafter. The integral can be nicely evaluated using the similarity, shift, and convolution theorems[1].

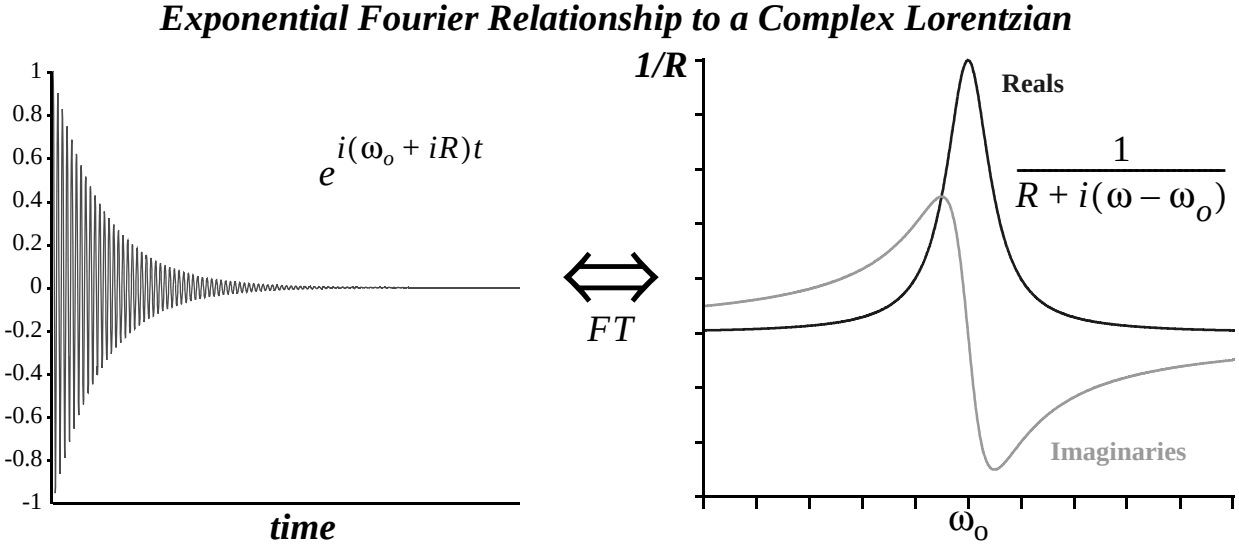


Figure C Depiction of a complex exponential. Only the reals of the exponential are plotted[2].

Some final comments are in order before we move to the next section, since this is often NOT the way things are derived in magnetic resonance. Often presentations just show a real Lorentzian as the result of the Fourier transformation on a exponential. Here are the details: Fourier transformation of a symmetrized decaying exponential from (-∞,∞) produces a ***real*** Lorentzian. Fourier transformation of the decaying exponential from [0, ∞) results in ***complex*** Lorentzians. One can use simple reasoning to determine why this is the case. The FT from ***(-∞,∞)*** (after time scaling and frequency shifting) involved a real symmetric function, the symmetrized exponential. Fourier transformation of a real symmetric function will produce a real symmetric function, the real Lorentzian. The FT on the unsymmetrized function from ***[0, ∞)*** can be visualized in terms of the function being decomposed into a real symmetric part and a real anti-symmetric part. The former produces the real Lorentzian component whereas the real anti-symmetric part produces the imaginary antisymmetric Lorentzian component.

1. See “The Fourier Transform and It’s Applications”, R.N. Bracewell, 2nd. Edition, McGraw-Hill Book Company, New York, 1978. See page 122 for the convolution and similarity theorems. The similarity and shift theorems are also know as frequency shifting and time scaling. These are discussed in Bracewell and in “The Fast Fourier Transform”, E.O. Brigham, Prentice-Hall, New Jersey, 1974.
2. Plot of exponential made with Lorentz1.cc on page 77

4.6.5 Differential Exponential

Differential exponential functions are also stumbled across in magnetic resonance, for instance in ESR where experiments are performed in CW mode[1]. The differential, as defined in GAMMA, is obtained by differentiation of the **Fourier integrand** of the exponential function with **respect to frequency**.

$$\begin{matrix} Fourier \\ Integrand \end{matrix} = FI = H(t)e^{i(\omega_o + iR)t}e^{-i\omega t}$$

$$\frac{d}{d\omega}[FI] = \frac{d}{d\omega}[H(t)e^{i(\omega_o + iR)t}e^{-i\omega t}] = -itH(t)e^{i(\omega_o + iR)t}e^{-i\omega t}$$

$$\frac{d}{d\omega}[FI] = H(t)[-ite^{i(\omega_o + iR)t}]e^{-i\omega t}$$

The differential exponential desired is

$$DE(t) = -ite^{i(\omega_o + iR)t} \tag{4-4}$$

A Typical Complex Differential Exponential

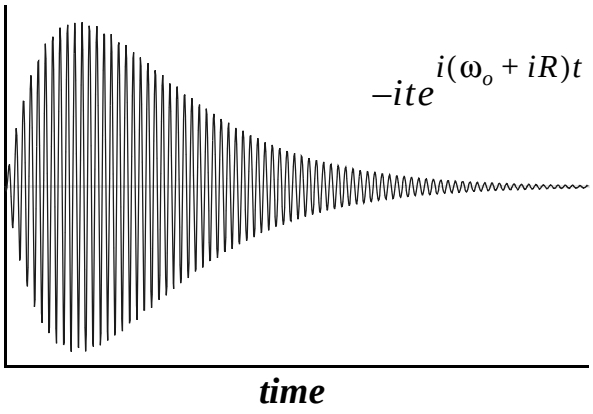


Figure D Depiction of a complex exponential. Only the reals are plotted[2].

Although the derivation above is a bit odd, the reason we use this definition is the differential exponential that we seek should be the Fourier transform pair of the differential Lorentzian - in this case the true result of differentiation of a complex Lorentzian with respect to frequency.

1. Actually, differential Lorentzians are encountered. These Lorentzians are GAMMA's "differential" exponentials when represented in the time domain.
2. Plot created by the program Exponent2.cc on page 77

4.6.6 Differential Lorentzian Relationship

Thus

$$DE(t) = -ite^{i(\omega_o + iR)t} \Leftrightarrow \frac{-i}{[R + i(\omega - \omega_o)]^2} = L'(\omega) = \frac{d}{d\omega}L(\omega) \qquad (4\text{-}5)$$

where $\Leftrightarrow$ implies a Fourier transform pair.

Differential Exponential Fourier Relationship

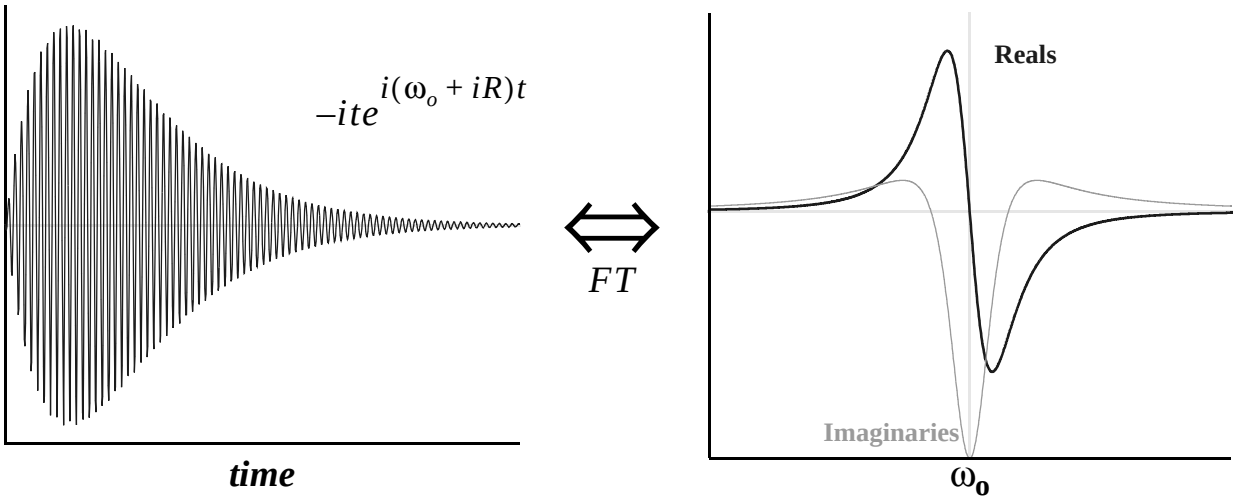


Figure E Depiction of a complex differntial exponential and its Fourier transform pair, a complex differential Lorentzian. Only the reals of the exponential are plotted[1].

The derivation of this relationship is given in the GAMMA documentation of the Lorentzian module. However, the relationship can be deemed true by invoking the Fourier differentiation theorem which says that if

$$h(t) \Leftrightarrow H(\omega)$$

then

$$-ith(t) \Leftrightarrow \frac{d}{d\omega}H(\omega)$$

1. Plot of exponential made with Exponent0.cc on page 76

4.6.7 Discrete Exponential

As a computer generated exponential function is discrete, we shall rewrite the function as a vector whose elements are given by

$$\langle E|k\rangle \;=\; e^{i(\omega_o + iR)t_k} \;=\; e^{i(\omega_o + i/T_2)t_k} \tag{4-6}$$

The vector will be assumed to contain ***N*** points which are indexed by the variable ***k***, ***k*** $\in$ ***[0,N)***. The time at point ***k*** will be given by t_k where

$$t_k \;=\; k \times t_{inc} \tag{4-7}$$

The value ω_o is the oscilation frequency of the exponential and ***R*** reflects the decay rate. The discrete exponential is depicted in the following figure with its analog form overlaid.

A Discrete Exponential

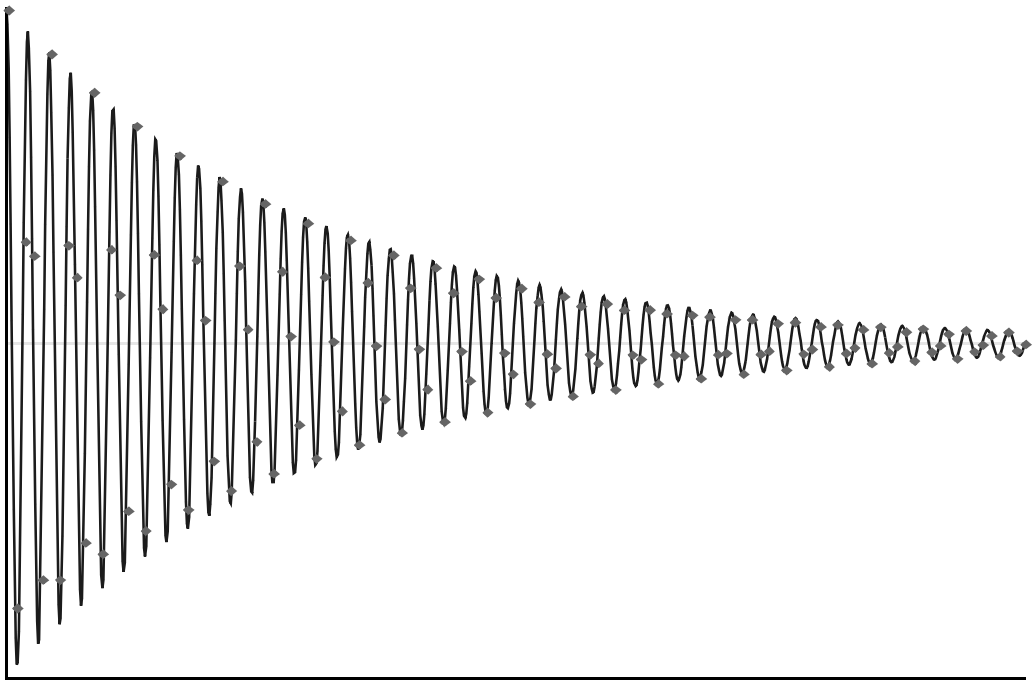


Figure F Depiction of the discrete Exponential function (points) overlaid on the analog form[1].

As is evident in the previous figure, if too few points are used in the discrete exponential then the function will be poorly characterized. This is related to the Nyquist frequency in Fourier transforms. In particular, the minimum number of points to use will put two point per cycle in the waveform. For a good discrete representation, > 2 points should occur per cycle.

1. Plot of exponential made with Exponent1.cc on page 76

4.6.8 Exponential Cutoffs

A final topic of consderation is the when the exponential falls below a specific cutoff value. This can be important in cases where many points of the exponential are below a set cutoff (smaller than computational roundoff for example) and explicit computation should be avoided to save time. For example, the rapidly decaying exponential in the next plot has the vast majority of its points essentially being zero. It would be a time savings to avoid calculating such points, instead just setting them to zero.

A Rapidly Decaying Exponential

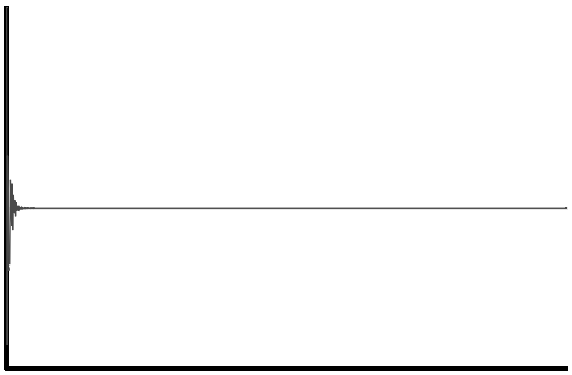


Figure G Depiction of a rapidly decaying complex exponential. Only the reals are plotted[1].

To avoid calculating these "zero" points we need to know where the exponential intensity falls below some set cutoff value or to within a % of the maximum value. We have previously seen that the magnitude of the complex exponential is given by

$$\|Exp(t)\| = e^{-Rt}$$

the maximum amplitude being 1. This value will reach a set value, ***cutoff***, when

$$cutoff = e^{-Rt_{cutoff}} \qquad t_{cutoff} = -\frac{\log(cutoff)}{R}$$

Note that this time is independent of the oscillation frequency but very much dependent upon the decay rate ***R***. Since we will work with a discrete exponential function, we will want to know at what point we this time t_{cutoff} corresponds to.

$$k_{cutoff} \times t_{inc} = t_{k,\,cutoff} \geq t_{cutoff} = -\frac{\log(cutoff)}{R}$$

Here $cutoff \in [0,1]$. Note that as cutoff approaches 0 the point ***k*** approches infinity and as the cutoff appraches 1 the point k approaches 0.

1. Plot created by the program Exponent2.cc on page 77

4.6.9 Exponential Equations

In this section we regroup the applicable equations regarding ***Exponential*** functions.

Exponential Equations

Analog Complex Exponential

$$Exp(t) = e^{i(\omega_o + iR)t} = e^{i(\omega_o + i/T_2)t}$$

Exponential Fourier Transform

$$L(\omega) = \frac{1}{R + i(\omega - \omega_o)}$$

Discrete Complex Exponential

$$\langle E|k\rangle = e^{i(\omega_o + iR)t_k} = e^{i(\omega_o + i/T_2)t_k}$$

Analog Differential Exponential

$$DE(t) = -ite^{i(\omega_o + iR)t}$$

Differential Exponential Fourier Transform

$$L'(\omega) = \frac{d}{d\omega}L(\omega) = \frac{-i}{[R + i(\omega - \omega_o)]^2}$$

4.7 Examples and Tests

4.7.1 Exponential Fourier Transform

As a first example we shall generate an exponential and then apply a Fourier transform to produce a Lorentzian. The generated Lorentzian will then be compared with a Lorentzian generated by GAMMA's Lorentzian module. There should be a 1 to 1 match up between the two functions when there is sufficient digitization and the spectral width is adequate.

Note that the FFT generated Lorentzian can suffer from the typical artifacts from discretization. This will result from the following problems in the time domain: baseline correction, truncation, undersampling, etc. These manifest themselves in the frequency domain as the following: spike at zero, non-zero Lorentzian baseline, frequency fold over, sinc behavior.

Exponential FFT To Generate A Lorentzian

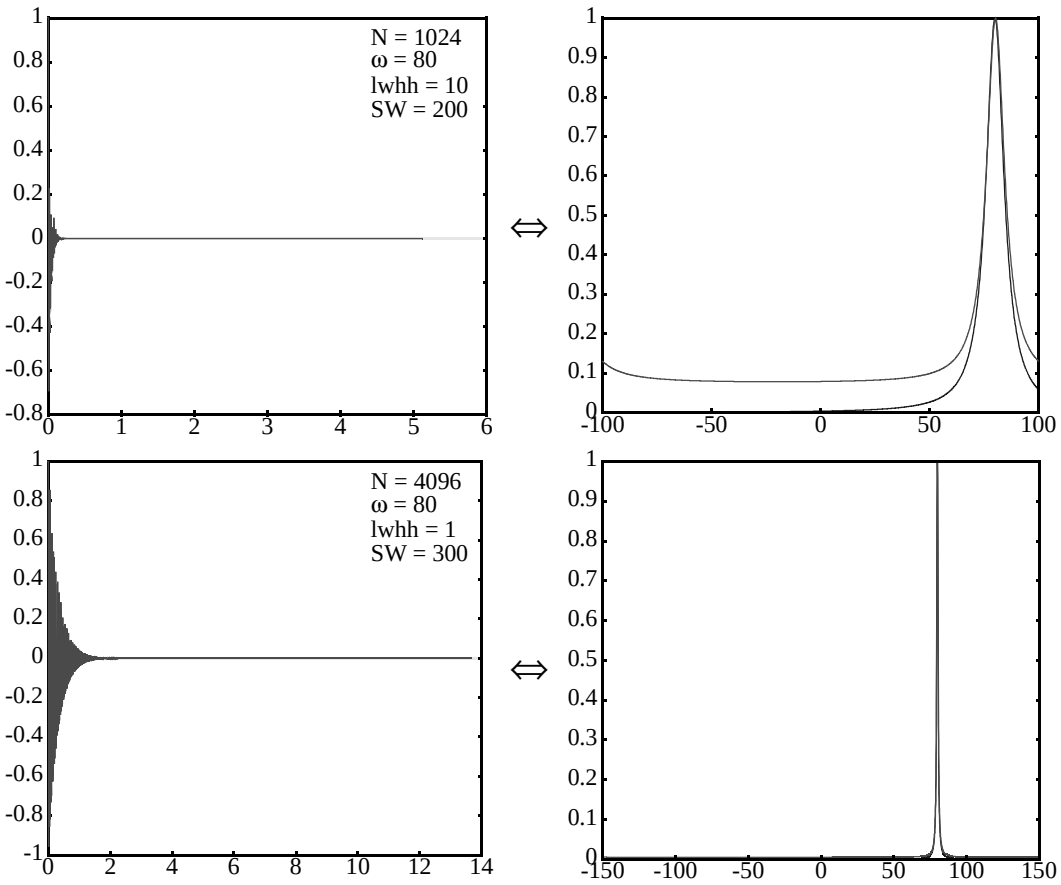


Figure H Demonstration of an FFT applied on a complex exponential. The left side are the time domain spectra (exponentials) and on the right are their Fourier transform pairs (Lorentzians). For comparison, a perfect Lorentzian is also plotted[1] (blue).

1. Plot created by the program xxxExponentT1.cc on xxxpage 77

4.7.2 Apodization

For our next example we wish to apply an “apodization”, i.e. force our time dependent function spanning [0, t] to exponentially decay at a specified rate (or equivalently with a specified time constant or with a rate corresponding to a specific Lorentzian linewidth).

withexample we shall generate an exponential and then apply a Fourier transform to produce a Lorentzian. The generated Lorentzian will then be compared with a Lorentzian generated by GAMMA’s Lorentzian module. There should be a 1 to 1 match up between the two functions when there is sufficient digitization and the spectral width is adequate.

4.8 Programs and Input

Exponent0.cc

```
/* Exponent0.cc ***************************************************************************
**                                                                                          **
**                GAMMA Exponential Generation Program                                      **
**                                                                                          **
** This program uses the GAMMA Exponential function module to build and                    **
** plot of simple complex Exponential function. The output will be                          **
** be interactive using Gnuplot.                                                            **
**                                                                                          **
** Author:    S.A. Smith                                                                    **
** Date:      7/21/97                                                                       **
** Copyright: S.A. Smith, September 1997                                                    **
**                                                                                          **
*********************************************************************************************/

#include <gamma.h>                              // Include GAMMA

main (int argc, char* argv[])
 {
 int qn=1;                                      // Query index
 int npts;
 query_parameter(argc, argv, qn++,              // Ask for block size
          "\n\n\tNumber of Points? ", npts);
 double R;                                      // For decay rate
 query_parameter(argc, argv, qn++,              // Ask for decay rate
                "\n\n\tDecay Rate? ", R);
 double W;                                      // For frequency
 query_parameter(argc, argv, qn++,              // Ask for frequency
                 "\n\n\tFrequency? ", W);
 row_vector CE = Exponential(2*npts,W,R);
 GP_1D("exp.asc", CE, 0);                       // Exponential out, GP ASCII
 GP_1Dplot("exp.gnu","exp.asc");                // Plot reals of Exponential
 cout << "\n\n";                                // Keep the screen nice
 }
```

Exponent1.cc

```
/* Exponent1.cc ***************************************************************************
**                                                                                          **
**                GAMMA Exponential Generation Program                                      **
**                                                                                          **
** This program uses the GAMMA Exponential function module to build and                    **
** plot of simple complex Exponential function. The output will be                          **
** be interactive using Gnuplot.                                                            **
**                                                                                          **
** The difference between this program and Exponent0.cc is that this                        **
** will also sketch in the value at time 1/R=T2 and uses an alternative                     **
** exponential function call.                                                               **
**                                                                                          **
**                                                                                          **
** Author:    S.A. Smith                                                                    **
** Date:      7/21/97                                                                       **
** Copyright: S.A. Smith, September 1997                                                    **
**                                                                                          **
*********************************************************************************************/

#include <gamma.h>// Include GAMMA

main (int argc, char* argv[])
 {
 int qn=1;                                      // Query index
 int npts;
 query_parameter(argc, argv, qn++,              // Ask for block size
          "\n\n\tNumber of Points? ", npts);
 double time;
 query_parameter(argc, argv, qn++,              // Ask for total time
           "\n\n\tTotal Time? ", time);
 double T2;                                     // For decay rate
 query_parameter(argc, argv, qn++,              // Ask for decay time
            "\n\n\tT2 Time? ", T2);
 double W;                                      // For frequency
 query_parameter(argc, argv, qn++,              // Ask for frequency
                 "\n\n\tFrequency? ", W);
 row_vector CE = Exponential(npts,time,W,1/T2);
 GP_1D("exp.asc", CE, 0, 0, time);              // Exponential out, GP ASCII
 row_vector EvalatT(3);
 EvalatT(0) = complex(0, 1/exp(1.0));
 EvalatT(1) = complex(T2, 1/exp(1.0));
 EvalatT(2) = complex(T2, 0);
 GP_xy("ExpatT.asc", EvalatT);                  // Lorentzian reals, GP ASCII
 String plots[2];
 plots[0] = "ExpatT.asc";
 plots[1] = "exp.asc";
 GP_1Dplot("exp.gnu", 2, plots);
 cout << "\n\n";                                // Keep the screen nice
 }
```

Lorentz1.cc

```
/* Lorentz1.cc ********************************************************************
**                                                                                  **
**         GAMMA Differential Lorentzian Generation Program                         **
**                                                                                  **
** This program uses the GAMMA Lorentzian function module to build and             **
** plot a simple differential Lorentzian function. The output will be              **
** interactive using Gnuplot. Both reals & imaginaries are plotted.                **
**                                                                                  **
** Author:    S.A. Smith                                                            **
** Date:      7/21/97                                                               **
** Copyright: S.A. Smith, September 1997                                            **
**                                                                                  **
************************************************************************************
/

#include <gamma.h>                              // Include GAMMA

main (int argc, char* argv[])
 {
 int qn=1;                                      // Query index
 int npts = 4096;                               // Set block size
 query_parameter(argc, argv, qn++,              // Ask for block size
        "\n\n\tNumber of Points? ", npts);

 double lwhh;                                   // For linewidth
 query_parameter(argc, argv, qn++,              // Ask for linewidth
     "\n\n\tHalf-height Linewidth? ", lwhh);
 double R = lwhh/2;                             // Set lwhh (2*R)

 double Wst, Wfi;
 query_parameter(argc, argv, qn++,              // Ask for start frequency
         "\n\n\tInitial Frequency? ", Wst);
 query_parameter(argc, argv, qn++,              // Ask for final frequency
         "\n\n\tFinal Frequency? ", Wfi);
 double W;                                      // The offset frequency
 query_parameter(argc, argv, qn++,              // Ask for offset
           "\n\n\tPeak Frequency? ", W);
 row_vector CL = DLorentzian(npts,Wst,Wfi,W,R);// Here is differential Lorentzian
 GP_1D("CR.asc", CL, 0, Wst, Wfi);              // DLorentzian reals, GP ASCII
 GP_1Dplot("CR.gnu","CR.asc");                  // Plot reals of DLorentzian
 GP_1D("CI.asc", CL, -1, Wst, Wfi);             // DLorentzian imaginaries, GP
ASCII
 GP_1Dplot("CI.gnu","CI.asc");                  // Plot imaginaries of DLorentzian
 cout << "\n\n";                                // Keep the screen nice
 }
```

Exponent2.cc

```
/* Exponent2.cc *******************************************************************
**                                                                                  **
**         GAMMA Differential Exponential Generation Program                        **
**                                                                                  **
** This program uses the GAMMA Exponential function module to build and            **
** plot simple differential Exponential functions. The output will                 **
** be interactive using Gnuplot.                                                    **
**                                                                                  **
** Author:    S.A. Smith                                                            **
** Date:      7/21/97                                                               **
** Copyright: S.A. Smith, September 1997                                            **
**                                                                                  **
************************************************************************************/

#include <gamma.h>                              // Include GAMMA

main (int argc, char* argv[])
 {
 int qn=1;                                      // Query index
 int npts;
 query_parameter(argc, argv, qn++,              // Ask for block size
         "\n\n\tNumber of Points? ", npts);
 double time;
 query_parameter(argc, argv, qn++,              // Ask for block size
             "\n\n\tTime Span? ", time);
 double R;
 query_parameter(argc, argv, qn++,              // Ask for span (hz)
              "\n\n\tDecay Rate? ", R);
 double W;
 query_parameter(argc, argv, qn++,              // Ask for span (hz)
               "\n\n\tFrequency? ", W);
 double tinc = time/double(npts-1);
 row_vector CE = DExponential(npts,tinc,W,R);
 GP_1D("DRexp.asc", CE, 0);                     // Exponential out, GP ASCII
 GP_1Dplot("DRexp.gnu","DRexp.asc");            // Plot reals of Exponential
 GP_1D("DIexp.asc", CE, -1);                    // Exponential out, GP ASCII
 GP_1Dplot("DIexp.gnu","DIexp.asc");            // Plot reals of Exponential
 cout << "\n\n";                                // Keep the screen nice
 }
```

5 Lorentzian

5.1 Overview

The ***Lorentzian*** module contains functions pertaining to the use of Lorentzian functions.

5.2 Available Functions

5.3 Description Sections

5.4 Lorentzian Figures

5.5 Lorentzian Programs

5.6 Lorentzian Functions

5.6.1 Lorentzian

Usage:

#include <Lorentzian.h>

complex Lorentzian(double Weval, double W, double R)

row_vector Lorentzian(int npts, double Wi, double Wf, double W, double R)

row_vector Lorentzian(int npts, double W, double R, int max1=0)

Description:

The function ***Lorentzian*** is obtain Lorentzian functions and function values.

1. Lorentzian(double, double, double) - Returns the complex Lorentzian value at the frequency **Weval** for the Lorentzian centered about frequency **W** and having linewidth 2**R**.
2. Lorentzian(int npts, double Wi, double Wf, double W, double R) - Returns a vector of **npts** containing a complex Lorentzian spanning the frequencies [**Wi**, **Wf**], centered about frequency **W** and linewidth 2**R**.
3. Lorentzian(int npts, double W, double R, int max1) - Returns a vector of **npts** containing a complex Lorentzian centered about point **W** and with linewidth 2**R** points. The Lorentzian will have a maximum magnitude of 1 unless **max1** is set non-zero whereupon it will have a max. magnitude of **R**.

Note that frequency and linewidth units in the first two functions are arbitrary but must be consistent. In the latter call the input units of **W** and **R** must be in points.

Return Value:

Either complex or row_vector.

Examples:

```
#include <gamma.h>
main()
  {
  complex z = Lorentzian(10., 30., 25.);     // Lorentzian @ 10 Hz where max = 30 Hz & lw = 50 Hz.
  row_vector L = Lorentzian(2000, -100., 300., 25.);; // Set pt equal to pt1.
  L = Lorentzian(2000, 500, 300);
  }
```

See Also: =

5.6.2 Lorentzian

Usage:

#include <Lorentzian.h>

void Lorentzian(int& ilo, int& ihi, R, W, w0, winc, npts, cutoff)

Description:

The function ***Lorentzian*** is used to create a 1-dimensional array containing a discrete Lorentzian function.

1. Acquire1D() - Creates an “empty” NULL acquire1D. Can be later filled by an assignment.
2. Acquire1D(gen_op &Op, gen_op &H, double dt) - Called with the operator for which the expectation

Return Value:

Void.

Examples:

See Also: =

5.6.3 Lorentz_cut

Usage:

void Lorentz_cut(int& ilo, int& ihi, double R, double W, double w0, double winc, int npts, double cutoff)
void Lorentz_cut(int* ilo, int* ihi, matrix& mx, double w0, double winc, int npts, double cutoff=1.e-4)

Description:

The function ***Lorentz_cut*** is used to determine the points outside of which a defined Lorentzian will have an intensity below a set percentage of its maximum. The desired Lorentzian(s) will begin at the frequency w0, have the frequency between points of winc, and have npts. The intensity cutoff is a decimal percent of the maximum allowed, cutoff. cutoff points ilo and ihi between which the Lorentzian defined by. The Lorentzian is taken to have a rate R, and a frequency W with initial frequency of w0 and frequency increment. The number of points in the Lorentzian is taken to be npts.

1. Lorentz_cut(int& ilo, int& ihi, double R, double W, double w0, double winc, int npts, double cutoff) - The Lorentzian rate and frequency are specified by the arguments R and W respectively. The integer values of ilo and ihi will be set accordingly.
2. Lorentz_cut(int& ilo, int& ihi, double R, double W, double w0, double winc, int npts, double cutoff) -

Return Value:

void

Examples:

See Also: =

5.6.4 Lorentz_int

Usage:

void Lorentz_int(int* Lint, matrix& mxi, double winc, int M=5)

Description:

The function ***Lorentz_int*** is used to determine whether or not a discrete Lorentzian should use integrated intensities or not. This determination is based on the minimum number of points allowed to span the Lorentzian linewidth at half-height, the number of points given by the argument ***M***. This criteria is mathematically given by

$$2R > M \; x \; \omega_{inc}$$

where other required values are the rate ***R*** and the frequency increment between points, ω_{inc}.

1. Lorentz_int(int& Lint, double R, double winc, int M=5) - The Lorentzian rate is taken to be ***R*** and the integration flag is set as ***Lint***.
2. Lorentz_int(int* Lint, matrix& mx, double winc, int M=5) - This function handles multiple Lorentzians whose rates are given in the real part of the first column of ***mx***. The number of rows of ***mx*** will set the

number of Lorentzians treated. The dimension of ***Lint*** must match the rows of ***mx*** and ***Lint*** will be filled with the integration flags for the Lorentzians.

Return Value:

Void.

Examples:

See Also: =

5.6.5 ask_Lorentzian

Usage:

void ask_Lorentzian(int argc, char* argv[], int& qn, int& N, double& wst, double& wfi, double& W, double & R, double& fact, int& pplw)

Description:

The function ***ask_Lorentzian*** is used for interactive programs. The function will query the user for parameters which define a Lorentzian function. These parameters are the number of points characterizing the function: ***N***, the initial and final starting frequencies: ***wst*** and ***wfi*** respectively, the offset: ***W*** (where peak maximum is), and the linewidth: ***R***. determine the points outside of which a defined Lorentzian will have an intensity below a set percentage of its maximum.

If the array ***argv*** of size ***argc*** contains the requested parametersstarting at position ***qn***, the values will be taken from the array rather than requested. The value of qn will be incremented by 6.

Return Value:

void

Example:

See Also: =

5.6.6 read_Lorentzian

Usage:

void read_Lorentzian(const string& filein, int argc, int& N, double& wst, double& wfi, double& W, double & R, double& fact, int& pplw, int idx=-1)

Description:

The function ***read_Lorentzian*** will read in parameters which define a Lorentzian function from an external ASCII file ***filein***. These parameters are the number of points characterizing the function: ***N***, the initial and final starting frequencies: ***wst*** and ***wfi*** respectively, the offset: ***W*** (where peak maximum is), and the linewidth: ***R***. determine the points outside of which a defined Lorentzian will have an intensity below a set percentage of its maximum.

If the value if idx in set to other than the default, the parameter names s are taken to be ***argv*** of size ***argc*** contains the requested parametersstarting at position ***qn***, the values will be taken from the array rather than requested. The value of qn will be incremented by 6.

Return Value:

void

Example:

See Also: =

5.6.7 DLorentzian

Usage:

void ask_Lorentzian(int argc, char* argv[], int& qn, int& N, double& wst, double& wfi, double& W, double & R, double& fact, int& pplw)

Description:

The function ***DLorentzian*** is used generate differential Lorentzian functions. The are mostly applicable to EPR computations. function will query the user for parameters which define a Lorentzian function. These parameters are the number of points characterizing the function: ***N***, the initial and final starting frequencies: ***wst*** and ***wfi*** respectively, the offset: ***W*** (where peak maximum is), and the linewidth: ***R***. determine the points outside of which a defined Lorentzian will have an intensity below a set percentage of its maximum.

If the array ***argv*** of size ***argc*** contains the requested parametersstarting at position ***qn***, the values will be taken from the array rather than requested. The value of qn will be incremented by 6.

Return Value:

void

Example:

See Also: =

5.7 Description

5.7.1 Overview

Lorentzians are commonly found in magnetic resonance because they are related to driven oscillators. Damped oscillations (decaying complex exponentials in time) are Lorentzian functions when displayed in the frequency domain (via a Fourier transform of the exponential). The ***Lorentzian*** module contains functions pertaining to the use of Lorentzian functions.

5.7.2 Analog Lorentzian

The general form of a complex Lorentzian function, as defined in GAMMA, is given by

$$L(\omega) = \frac{R - i(\omega - \omega_o)}{R^2 + (\omega - \omega_o)^2} = \frac{1}{R + i(\omega - \omega_o)} \tag{5-1}$$

Here, ω_o will be the frequency at the center of the Lorentzian and R is a rate which relates to the half-height linewidth. For mathematical purposes it is best to keep both of these values in radians/sec (which avoids having factors of π in equations), however Hertz is generally preferred in MR work. We will assume the former herein and point out the conversions required (if any) to switch into other units.

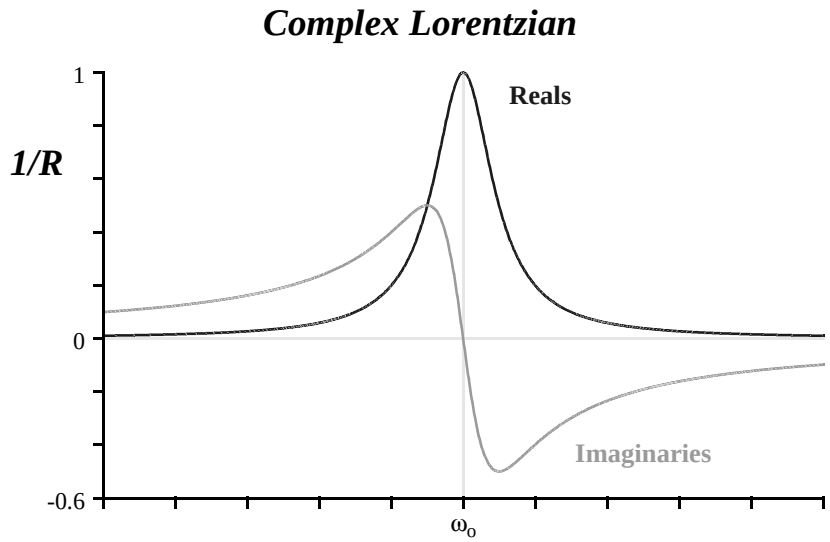


Figure A Depiction of a complex Lorentzian, reals and imaginaries plotted separately[1].

The maximum of the real part of Lorentzian spectrum is found at $\omega=\omega_0$ and has amplitude of ***1/R***. Since ***R*** is related to the linewidth, for larger ***R*** (broader Lorentzian) the maximum height is smaller because the integral over frequency remains constant.

1. Plot made with Lorentz0.cc on page 95

5.7.3 Lorentzian Half-Height Linewidth

A common way to characterize ***R*** is to give the half-height linewidth of the Lorentzian, ***lwhh***.

The Lorentzian Half-Height Linewidth

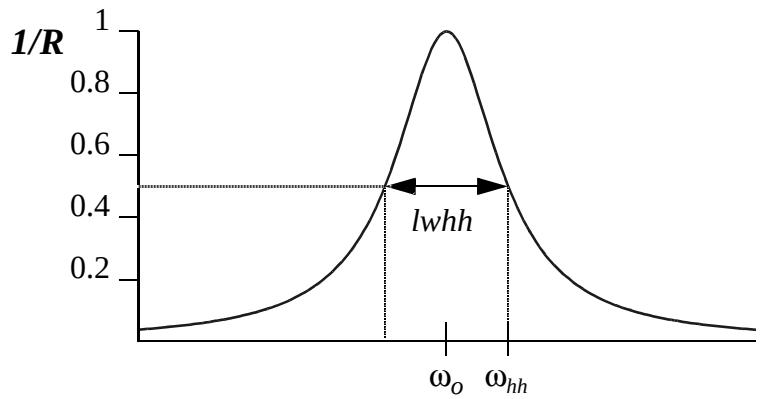


Figure B Depiction of real Lorentzian half-height linewidth[1].

This is readily derived by determining where the real part of the function reaches half of its maximum intensity.

$$\frac{1}{2} = \frac{L(\omega_{hh})}{L(\omega_{max})} = \frac{R^2 + (\omega_o - \omega_o)^2}{R^2 + (\omega_{hh} - \omega_o)^2} = \frac{R^2}{R^2 + (\omega_{hh} - \omega_o)^2}$$

$$R = \pm(\omega_{hh} - \omega_o) \qquad \omega_{hh} = \omega_o \pm R$$

Evidently, the real Lorentzian reaches it half-height at ***+/- R*** from it's center. The total distance between these two frequencies is defined as the half height linewidth (see previous figure).

$$\mathbf{lwhh = 2R} \qquad \textbf{(5-2)}$$

1. Plot made with Lorentz2.cc on page 95

5.7.4 Fourier Relationship to Complex Exponentials

Lorentzians are so commonly encountered because they are the frequency form of driven oscillators. The time domain form of a driven oscillator is a complex decaying exponential, and the two are related *via* a Fourier transformation.

Consider the decaying complex exponential we are likely to encounter in experimental work. It begins at time ***t=0***, oscillates at frequency ω_o, and decays at some rate ***R***. An example is given in the following figure with the exponential functional form.

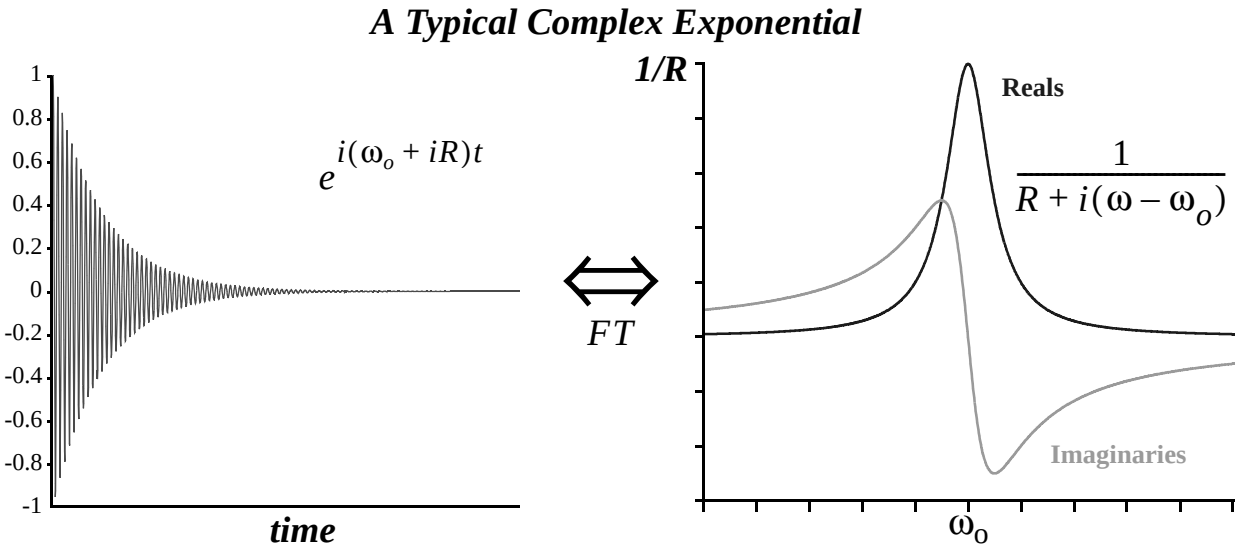


Figure C Depiction of a complex exponential. Only the reals of the exponential are plotted[1].

The Fourier transform of an exponential is given by

$$F(\omega) = \int_{-\infty}^{\infty} H(t) e^{i(\omega_o + iR)t} e^{-i\omega t} dt$$

Since the exponential does not span the Fourier integral range of ±∞ we have made use of the Heavyside function, ***H(t)***, which is zero until it reaches ***t=0*** and 1 thereafter. The integral can be nicely evaluated using the similarity, shift, and convolution theorems[2]. The integral is initially expanded by separation of the real and imaginary terms in the exponentials.

1. Plot of exponential made with Exponent0.cc on page 95
2. See “The Fourier Transform and It’s Applications”, R.N. Bracewell, 2nd. Edition, McGraw-Hill Book Company, New York, 1978. See page 122 for the convolution and similarity theorems. The similarity and shift theorems are also know as frequency shifting and time scaling. These are discussed in Bracewell and in “The Fast Fourier Transform”, E.O. Brigham, Prentice-Hall, New Jersey, 1974.

$$F(\omega) = \int_{-\infty}^{\infty} H(t)e^{-Rt}e^{-i(\omega-\omega_0)t}dt$$

The frequency shift theorem (or frequency shifting) of Fourier transforms states that a function which has been frequency shifted has the Fourier transform of the unshifted function will all frequencies replaced by the shifted frequency. So we can easily rewrite our integral as

$$F(\omega) = F(\omega') = \int_{-\infty}^{\infty} H(t)e^{-Rt}e^{-i\omega' t}dt \qquad \omega' = \omega - \omega_0$$

The similarity theorem (or time scaling) of Fourier transforms states that a function which has been time scaled has the Fourier transform of the un-scaled function with all frequencies divided by the scaling factor as well as an inverse scaling of the overall transform. If we first multiply the Heavyside time axis by R (which has no effect on the function), we can directly employ this theorem.

$$F(\omega) = \int_{-\infty}^{\infty} H(t)e^{-Rt}e^{-i\omega' t}dt = \int_{-\infty}^{\infty} H(Rt)e^{-Rt}e^{-i\omega' t}dt$$

$$F(\omega) = \frac{F(\omega'')}{|R|} = \frac{1}{|R|}\int_{-\infty}^{\infty} H(t)e^{-t}e^{-i\omega'' t}dt \qquad \omega'' = \frac{\omega'}{R} = \frac{\omega-\omega_0}{R}$$

This is of course valid for $\boldsymbol{R\neq 0}$ only, but then time scaling isn't required if that is indeed the case. The situation when $\boldsymbol{R=0}$ will be discussed shortly. Furthermore, we won't have to worry about the absolute value on $\boldsymbol{R}$ because real rate's are non-negative numbers[1]. By a simple rearrangementwe can recognize the remaining Fourier integral as that of a real exponential.

$$|R|F(\omega) = F(\omega'') = \int_{-\infty}^{\infty} H(t)e^{-|t|}e^{-i\omega'' t}dt \qquad \omega'' = \frac{\omega-\omega_0}{R}$$

There is no loss in generality by switching to the absolute value in the exponential due the use of the Heavyside function.

This Fourier transform is well known[2].

$$|R|F(\omega) = F(\omega'') = \int_{-\infty}^{\infty} H(t)e^{-|t|}e^{-i\omega'' t}dt = \frac{1-i\omega''}{1+\omega''^2} \qquad \omega'' = \frac{\omega-\omega_0}{R}$$

1. The rates $\boldsymbol{R}$ should always be non-negative numbers in this treatment so that the exponentials do not blow up with time. They pair up with frequencies $-\omega_o$ which are components that will be shifted to ω_o from 0.
2. See "The Fourier Transform and It's Applications", R.N. Bracewell, 2nd. Edition, McGraw-Hill Book Company, New York, 1978. The shown transform is found on page 392.

a real rate[1] component $\boldsymbol{R}$; $\boldsymbol{z = R - i\omega_0}$.

Our final desired integral is then, using $\Delta\omega = \omega - \omega_0$,

$$F(\omega) = \frac{1}{|R|}F(\omega") = \frac{1}{|R|}\left[\frac{1 - i\Delta\omega/R}{1 + (\Delta\omega/R)^2}\right]$$

This has other forms that are more useful to this discussion. We shall assume a positive $\boldsymbol{R}$, i.e. a real rate (decaying exponential).

$$F(\omega) = \frac{1}{R}\left[\frac{1 - i\Delta\omega/R}{1 + (\Delta\omega/R)^2}\right] = \frac{R - i\Delta\omega}{R^2 + \Delta\omega^2} = \frac{e^{-i\,\mathrm{atan}\left(\frac{\Delta\omega}{R}\right)}}{\sqrt{R^2 + \Delta\omega^2}} = \frac{1}{R + i\Delta\omega}$$

You guessed it! This is none other than our Lorentzian function! So, the complex Lorentzian is the Fourier transform of a decaying exponential.

$$L(\omega) = \frac{1}{R + i\Delta\omega} = FT[e^{i(\omega_o + iR)t}] = \int_{-\infty}^{\infty} H(t)e^{-Rt}e^{-i(\omega - \omega_0)t}dt \qquad (5\text{-}3)$$

Some final comments are in order before we move to the next section, since this is often NOT the way things are derived in magnetic resonance. Typical MR presentations just show a real Lorentzian as the result of the Fourier transformation on an exponential. Here are the details: Fourier transformation of a symmetrized decaying exponential from (-∞,∞) produces a ***real*** Lorentzian. Fourier transformation of the decaying exponential from [0, ∞) results in ***complex*** Lorentzians. One can use simple reasoning to determine why this is the case. The FT from (-∞,∞) (after time scaling and frequency shifting) involved a real symmetric function, the symmetrized exponential. Fourier transformation of a real symmetric function will produce a real symmetric function, the real Lorentzian. The FT on the unsymmetrized function from [0, ∞) can be visualized in terms of the function being decomposed into a real symmetric part and a real anti-symmetric part. The former produces the real Lorentzian component whereas the real anti-symmetric part produces the imaginary antisymmetric Lorentzian component.

1. The rates $\boldsymbol{R}$ should always be non-negative numbers in this treatment so that the exponentials do not blow up with time. They pair up with frequencies $-\omega_o$ which are components that will be shifted to ω_o from 0.

5.7.5 Differential Lorentzian

Differential Lorentzian functions are also stumbled across in magnetic resonance, for instance in ESR where experiments are performed in CW mode. The differential, as defined in GAMMA, is obtained by differentiation of the Lorentzian function with respect to frequency.

$$L(\omega) = \frac{1}{R + i(\omega - \omega_o)} \qquad L'(\omega) = \frac{d}{d\omega}L(\omega) = \frac{-i}{[R + i(\omega - \omega_o)]^2} \tag{5-4}$$

Here, ω_o will be the frequency at the center of the Lorentzian and R is a rate which relates to the half-height linewidth. For mathematical purposes it is best to keep both of these values in radians/sec (which avoids having factors of π in equations), however Hertz is generally preferred in MR work. We will assume the former herein and point out the conversions required (if any) to switch into other units.

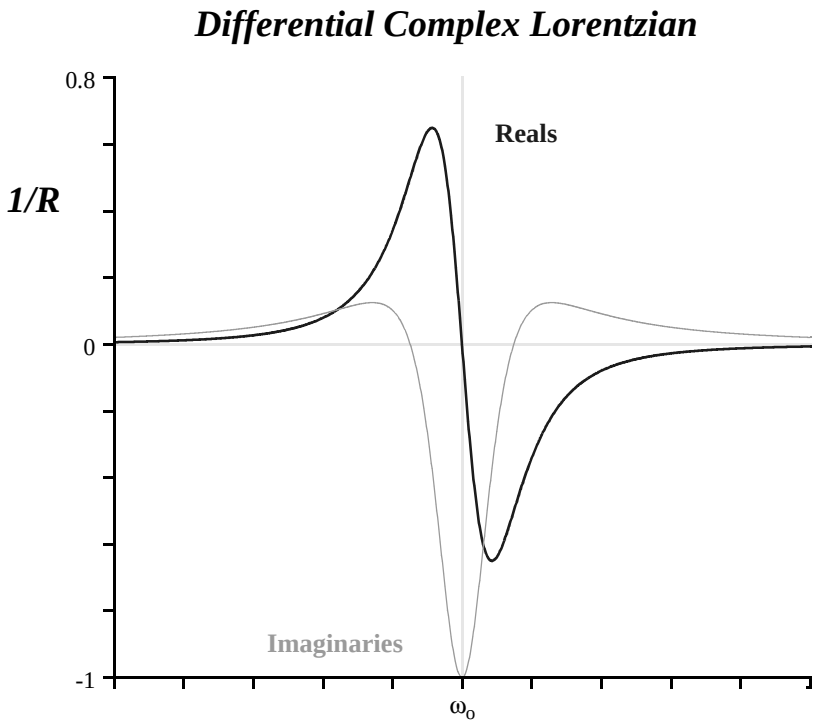


Figure D Differential complex Lorentzian, reals and imaginaries plotted separately[1].

Although they look quite similar, the real part of the differential Lorentzian is NOT equivalent to the imaginary part of the Lorentzian! Indeed, the lineshape of the differential form is much narrower that the dispersive part of the Lorentzian.

1. Figure produced with the program Lorentz0.cc on page 95.

5.7.6 Fourier Relationship to Differential Exponentials

Differential Lorentzians may also be related to a time domain domain function, a sort of “differential exponential”, through a Fourier transformation. In this instance we can direclty obtain the time domain function through use of a Fourier differentiation theorm. The theorem states that if f(t) has the transform pair F(ω) then -itf(t) is the transform pair of F’(ω).

The the theorem is easy to prove, and we will just show a quick derivation in the case of our Lorentzian.

$$
\begin{aligned}
L(\omega) &= \frac{1}{R + i\Delta\omega} = \int_{-\infty}^{\infty} H(t)e^{-Rt}e^{-i(\omega-\omega_0)t}dt = FT[e^{i(\omega_o + iR)t}] \\
L'(\omega) &= \frac{-i}{[R + i(\omega-\omega_o)]^2} = \frac{d}{d\omega}L(\omega) = \frac{d}{d\omega}\int_{-\infty}^{\infty} H(t)e^{-Rt}e^{-i(\omega-\omega_0)t}dt \\
L'(\omega) &= \int_{-\infty}^{\infty} \frac{d}{d\omega}[H(t)e^{-Rt}e^{-i(\omega-\omega_0)t}]dt = \int_{-\infty}^{\infty} H(t)e^{-Rt}\frac{d}{d\omega}[e^{-i(\omega-\omega_0)t}]dt \\
L'(\omega) &= \int_{-\infty}^{\infty} H(t)e^{-Rt}(-it)e^{-i(\omega-\omega_0)t}dt = FT[-ite^{i(\omega_o + iR)t}]
\end{aligned}
\tag{5-5}
$$

5.7.7 Discrete Lorentzian

As a computer generated Lorentzian function is discrete, we shall rewrite the function as a vector whose elements are given by

$$\langle L|l\rangle \;=\; \frac{R - i(\omega_l - \omega_o)}{R^2 + (\omega_l - \omega_o)^2} \;=\; \frac{R - i\Delta\omega_l}{R^2 + \Delta\omega_l^2} \;=\; \frac{1}{R + i\Delta\omega_l} \tag{5-6}$$

The vector will be assumed to contain $\boldsymbol{N}$ points which are indexed by the variable $\boldsymbol{l}$, $\boldsymbol{l} \in [0,N)$. The frequency at point $\boldsymbol{l}$ will be given by $\omega_{\boldsymbol{l}}$ where

$$\omega_l \;=\; \omega_0 + l \times \omega_{inc} \tag{5-7}$$

The value $\omega_{\boldsymbol{o}}$ is the frequency of the first point, $\boldsymbol{l=0}$, and the value $\omega_{\boldsymbol{inc}}$ the frequency difference between adjacent points. Note that $\omega_{\boldsymbol{0}} \neq \omega_{\boldsymbol{o}}$ in general! The discrete Lorentzian is depicted in the following figure with its analog form overlaid.

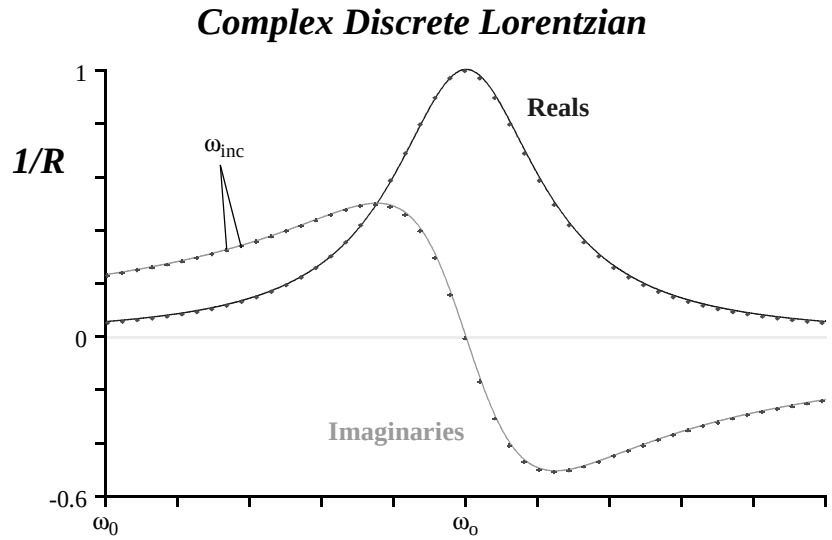


Figure E Depiction of a complex Lorentzian, reals and imaginaries plotted separately. Solid lines indicate the analog Lorentzian whereas the dots indicate the discrete points which characterize the function.

Although the analog Lorentzian requires only two parameters, $\boldsymbol{R}$ and $\omega_{\boldsymbol{o}}$, to fully specify the function, the discrete Lorentzian demands a relationship between the number of points and the point spread. Thus the specification of a discrete Lorentzian requires, in addition to $\boldsymbol{R}$ and $\omega_{\boldsymbol{o}}$, $\{\boldsymbol{N}, \omega_{\boldsymbol{o}}, \omega_{\boldsymbol{f}}\}$ where $\omega_{\boldsymbol{f}}$ is the frequency at the last point in the vector.

A problem can occur in discrete representation of a Lorentzian if the frequency increment $\omega_{\boldsymbol{inc}}$ is large relative to the Lorentzian linewidth, $\boldsymbol{2R}$. If such is the case, the peak maximum will be poorly sampled - perhaps resulting in a discrete spectrum which is essentially all zero's even though it's sampled range includes the maximum frequency $\omega_{\boldsymbol{o}}$.

5.7.8 Integrated Intensities

To circumvent the problem of under sampling it is often better to fill the discrete Lorentzian points with averaged integrated intensities rather than the values at specific frequencies. The necessary integrals, in analog form, are given by

$$\int \frac{1}{R^2 + \Delta\omega^2} d\Delta\omega = \frac{1}{R}\operatorname{atanh}\frac{\Delta\omega}{R} \qquad \int \frac{\Delta\omega_k}{R^2 + \Delta\omega^2} d\Delta\omega = \frac{1}{2}\log(R^2 + \Delta\omega^2) \tag{5-8}$$

What is required in this instance are averaged integrated intensities over the discrete function, the integral spanning $\omega_{\boldsymbol{l}} \pm \omega_{\boldsymbol{inc}}/2$. The appropriate equations are thus

$$\begin{aligned} Re\langle L|l\rangle &= \frac{1}{\omega_{inc}}\operatorname{atanh}\frac{\Delta\omega_l + \omega_{inc}/2}{R} - \operatorname{atanh}\frac{\Delta\omega_l - \omega_{inc}/2}{R} \\ Im\langle L|l\rangle &= \frac{-i}{2\omega_{inc}}\log\left(\frac{R^2 + (\Delta\omega_l + \omega_{inc}/2)^2}{R^2 + (\Delta\omega_l - \omega_{inc}/2)^2}\right) \end{aligned} \tag{5-9}$$

Obviously, if the discrete function is well sampled then integrated intensities are not necessary; in fact they are computationally slower to generate. A question which must be answered is thus when should the function be considered well sampled? There are three inter-connected values which must be used to determine the answer: 1.) The linewidth of the Lorentzian being generated. 2.) The frequency increment taken. and 3.) The minimum number of points needed to span the linewidth at half-height.

The linewidth at half-height is[1] given in inverted seconds as

$$\boldsymbol{lwhh = 2R} \tag{5-10}$$

If we assume that we require $\boldsymbol{M}$ points sampling the discrete Lorentzian and we know that each point itself covers $\omega_{\boldsymbol{inc}}$, then we must satisfy

$$\boldsymbol{2R > M\ x\ \omega_{inc}} \tag{5-11}$$

If this equation is satisfied they there is little reason to use integrated intensities. If it is not satisfied, it is probably better to use the integrated values so that the spectrum represents the true lineshape better.

5.8 Spectral Range

Consider the situation where a discrete Lorentzian is desired over a particular frequency range. It

1. In NMR spectroscopy the linewidth at half-height is often related to the spin-lattice relaxation rate $\boldsymbol{R}$ and the spin-lattice relaxation time T_2. We have used $\boldsymbol{R}$ as a constant parameter related to the Lorentzian width and indeed it is the relaxation rate. Thus, $\boldsymbol{R = R_2 = 1/}T_2$ and the linewidth, *in Hertz*, is $\boldsymbol{lwhh = R_2/\pi}$

may be that the Lorentzian intensity is effectively zero (within computer roundoff) for all or a large part of the computed function. It would be much more efficient to just set such elements to zero rather than computing them. To do so we would need to know where the Lorentzian intensity falls below some set cutoff value or to within a % of the maximum value.

The magnitude of the complex Lorentzian is given by

$$\left\| L_k(\omega) \right\| = \sqrt{\frac{R_k^2 + \Delta\omega^2}{[R_k^2 + \Delta\omega^2]^2}} = \sqrt{\frac{1}{R_k^2 + \Delta\omega_k^2}} \tag{5-12}$$

Evidently, the maximum amplitude is $\boldsymbol{1/R_k}$. We shall investigate where this magnitude falls below the value $\boldsymbol{cutoff/R_k}$ where $\boldsymbol{cutoff \in [0, 1]}$. For the discrete Lorentzian function

$$cutoff/R_k = \sqrt{\frac{1}{R_k^2 + \Delta\omega_{k,l}^2}}$$

which produces

$$\omega_l = \omega_k \pm R_k \sqrt{\frac{1}{cutoff^2} - 1} \tag{5-13}$$

The two frequencies ω_l above are the values at which the Lorentzian magnitude is equal to the selected value $\boldsymbol{cutoff/R_k}$. If cutoff is taken as the decimal form of the percentage of maximum amplitude desired calculated, then values outside of this range need not be computed.

Since we are dealing with a discrete function, the available frequencies can be directly related to the integer index $\boldsymbol{l}$. The previous equation becomes

$$\omega_0 + l \times \omega_{inc} = \omega_k \pm R_k \sqrt{\frac{1}{cutoff^2} - 1}$$

or equivalently

$$l = \frac{1}{\omega_{inc}} \left[\omega_k - \omega_0 \pm R_k \sqrt{\frac{1}{cutoff^2} - 1} \right] \tag{5-14}$$

Note that as ω_{inc} increases the number of points with magnitude within the cutoff will decrease. Also, there is a slight inconsistency in the discrete *versus* analog treatment. If $\boldsymbol{cutoff = 1}$ or ω_{inc} is large there may be one point in the range: $\boldsymbol{l = int[(\omega_k - \omega_0)/\omega_{inc}]}$. This should be expanded to the three nearest points to then there will be approximately one point which is within the cutoff. One the other hand if $\boldsymbol{cutoff = 0}$ then the range will span all possible points but this cannot be because of the implicit range being computed, $\boldsymbol{l \in [0, npts)}$.

5.9 Fourier Relationship

Lorentzians are commonly encountered because they are the frequency form of driven oscillators. The time domain form of a driven oscillator is a complex decaying exponential, and the two are related *via* a Fourier transformation.

$$F(\omega) = \int_{-\infty}^{\infty} H(t) e^{i(\omega_o + iR)t} e^{-i\omega t} dt$$

Of course, analog Fourier transforms (FT's) on theoretical equations do not exactly match up with discrete Fourier transforms (DFT's) on experimental data. This is partly due to the analog Fourier transform extending in time to span (-∞,∞), whereas experimental data acquired in time will normally span from some designated t=0 to some finite time $t=t_f$ and the frequency spectrum obtained from a discrete Fourier transform (DFT). These differences can be inconsequential under the appropriate conditions, one of which is to acquire experimental data points until their intensity is zero (if possible). If it is assumed that this is the case, one may safely replace t_f by t=∞ and that is what is done herein.

It will also be assumed either that the desired frequency spectrum will be the result of a FT of the data acquired from time 0 to infinite time. The Fourier integrals and a discussions of differences between the two possible treatments, 1.) The Fourier integral taken over the range (-∞,∞) with the function symmetric about $t=0$ and 2.) The "Fourier" integral is taken over the range [0, ∞), will be evaluated. It can be argued that the DFT makes it appear as if the acquired points were folded[1] about $t=0$, and that nothing drastic is lost or added by the treatment over the range (-∞,∞).

The "Fourier" transforms can all be written as transforms of the function f where $f=e^{-zt}$ and z is a complex number.

$$f = e^{-zt} \qquad I(\omega) = \int_{0}^{\infty} f e^{-i\omega t} dt \tag{5-15}$$

The integral is not formally a Fourier transform, but can be made to be so with multiplication by the Heavyside function, $H(t)$. This extends the integral range so it appears as well defined Fourier transform (and later allows for the use of the convolution theorem for its evaluation).

$$I(\omega) = \int_{0}^{\infty} f e^{-i\omega t} dt = \int_{-\infty}^{\infty} H(t) f e^{-i\omega t} dt \tag{5-16}$$

This integral can be nicely evaluated using the similarity, shift, and convolution theorems[2]. The integral is initially expanded by expressing the complex value z in terms of its imaginary frequency

1. See "The Fast Fourier Transform", E.O. Brigham, Prentice-Hall, Inc., New Jersey, 1974. The "apparent" folding over of sampled points when performing a DFT is graphically depicted on pages 92 and 95.

component ω_0 and a real rate[1] component $\boldsymbol{R}$; $\boldsymbol{z = R - i\omega_0}$

$$I(\omega) = \int_{-\infty}^{\infty} H(t)e^{-(R-i\omega_0)t}e^{-i\omega t}dt = \int_{-\infty}^{\infty} H(Rt)e^{-Rt}e^{-i(\omega-\omega_0)t}dt \tag{5-17}$$

The multiplication of the Heavyside time variable by $\boldsymbol{R}$ does not affect the integral.

The frequency shift theorem (or frequency shifting) of Fourier transforms states that a function which has been frequency shifted has the Fourier transform of the unshifted function will all frequencies replaced by the shifted frequency.

$$I'(\omega) = \int_{-\infty}^{\infty} H(Rt)e^{-Rt}e^{-i\omega t}dt \qquad I(\omega) = I'(\omega - \omega_0)$$

The similarity theorem (or time scaling) of Fourier transforms states that a function which has been time scaled has the Fourier transform of the un-scaled function with all frequencies divided by the scaling factor as well as an inverse scaling of the overall transform. This is valid for $\boldsymbol{R} \neq 0$ only!.

$$I''(\omega) = \int_{-\infty}^{\infty} H(t)e^{-|t|}e^{-i\omega t}dt \qquad I'(\omega) = \frac{1}{|R|}I''\left(\frac{\omega}{R}\right) \qquad I(\omega) = \frac{1}{|R|}I''\left(\frac{(\omega-\omega_0)}{R}\right)$$

There is no loss in generality by switching to the absolute value in the exponential due the use of the Heavyside function.The situation when $\boldsymbol{R}$=0 will be discussed shortly. This Fourier transform is well known[2].

$$I''(\omega) = \int_{-\infty}^{\infty} H(t)e^{-|t|}e^{-i\omega t}dt = \frac{1-i\omega}{1+\omega^2}$$

The desired integral is then, using $\Delta\omega = \omega - \omega_0$,

$$I(\omega) = \frac{1}{|R|}I''\left(\frac{(\omega-\omega_0)}{R}\right) = \frac{1}{|R|}\left(\frac{1-i\Delta\omega/R}{1+(\Delta\omega/R)^2}\right) \tag{5-18}$$

---

2. See “The Fourier Transform and It’s Applications”, R.N. Bracewell, 2nd. Edition, McGraw-Hill Book Company, New York, 1978. See page 122 for the convolution and similarity theorems. The similarity and shift theorems are also know as frequency shifting and time scaling. These are discussed in Bracewell and in “The Fast Fourier Transform”, E.O. Brigham, Prentice-Hall, New Jersey, 1974.

1. The rates $\boldsymbol{R}$ should always be non-negative numbers in this treatment so that the exponentials do not blow up with time. The pair up which frequencies $-\omega_0$ which are components that will be shifted to ω_0 from 0.

2. See “The Fourier Transform and It’s Applications”, R.N. Bracewell, 2nd. Edition, McGraw-Hill Book Company, New York, 1978. The shown transform is found on page 392.

5.10 Lorentzian Equation Set

In this section we regroup the applicable equations regarding ***Lorentzian*** functions.

Lorentzian Equations

Analog Lorentzian

$$L(\omega) = \frac{R - i(\omega - \omega_o)}{R^2 + (\omega - \omega_o)^2} = \frac{1}{R + i(\omega - \omega_o)}$$

Half-Height Linewidth

$$lwhh = 2R$$

Discrete Lorentzian

$$\langle L|l\rangle = \frac{R - i(\omega_l - \omega_o)}{R^2 + (\omega_l - \omega_o)^2} = \frac{1}{R + i(\omega_l - \omega_o)}$$

Fourier Transform Pair

$$\int_{-\infty}^{\infty} H(t)e^{-Rt}e^{-i(\omega - \omega_0)t}dt$$

$$L(\omega) \Longleftrightarrow e^{i(\omega_o + iR)t}$$

Analog Differential Lorentzian

$$DL(\omega) = \frac{-i}{[R + i(\omega - \omega_o)]^2}$$

Fourier Transform Pair

$$L(\omega) \Longleftrightarrow -i\omega e^{i(\omega_o + iR)t}$$

6 Window Functions

6.1 Overview

This module provides some primitive functions that are used in NMR data processing. They may be applied to calculated FID's produced by GAMMA simulations, used to construct shaped pulses, and anything else available to GAMMA row vectors. GAMMA does not attempt to support a full range of processing routines. For this there a several commercial packages available to do the job. However, some processing routines are highly useful in that they allow for a facile and direct evaluation of simple simulated data types.

Note that processing steps such as FFT's are inherent properties GAMMA's vectors and matrices. Furthermore, specific modules exist to handle functions that are useful beyond windowing such as exponentials and Lorentzians. Finally, several shaped pulses (e.g. Gaussian) have specific pulse modules which support them directly.

6.2 Available Functions

6.3 Function Discussions

6.4 Routines

6.4.1 exponential_multiply

Usage:

#include <Level1/WindowFct.h>
void exponential_multiply(row_vector &db, double em=-4);

Description:

This function is deprecated. There is now a entire exponential function module in GAMMA that has this and related functions. Users should see the related documentation and use one of the provided functions therein.

The function ***exponential_multiply*** multiplies the values in ***db*** with an exponential function:

$$db_i = db_i e^{em\frac{i}{size(db)}}$$

Therefore the first point is multiplied by 1 and the last by e^{em}.

Return Value:

Nothing. The vector db is modified.

Example:

```
#include <gamma.h>
main()
  {
  row_vector SPECTRUM;
  exponential_multiply (SPECTRUM);
  }
```

See Also:

6.4.2 sinc

Usage:

```
#include <Level1/WindowFct.h>
row_vector sinc(int size, int offset, int node);
```

Description:

The window function ***sinc*** returns a row vector containing a sinc function. The block returned has the size specified by the argument ***size***. The sinc function will have its maximum at the point specified by the argument ***offset*** and the first node of the function will be at ***node*** points away from the maximum.

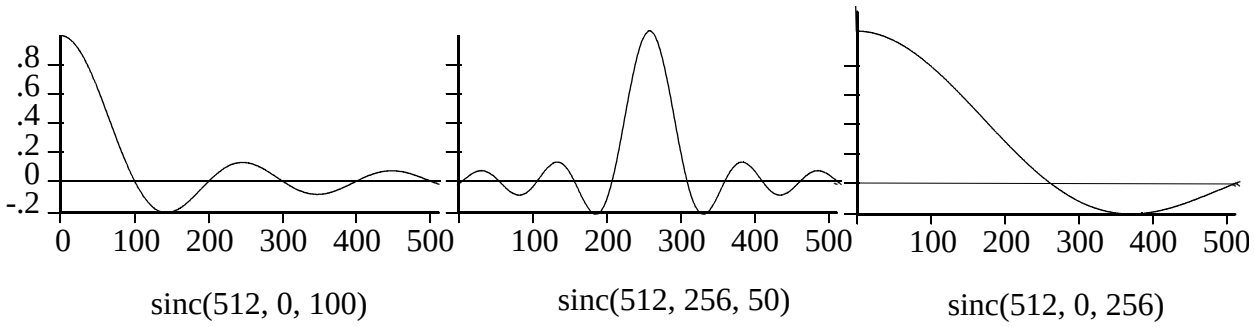
Return:

A row vector.

Examples:

```
#include <gamma.h>
main()
  {
  row_vector BLK(512);
  BLK = sinc(512, 0, 100);
  FM_1D ("sinc1.mif", BLK, 10, 5, 0, 511);
  BLK = sinc(512, 256, 50);
  FM_1D ("sinc2.mif", BLK, 10, 5, 0, 511);
  BLK = sinc(512, 0, 256);
  FM_1D ("sinc3.mif", BLK, 10, 5, 0, 511);
  }
```

In this example, the function sinc is called three times with different parameters. Each returned vector is then written to a FrameMaker file. These have been imported into this document and shown below after slight cosmetic adjustment.



See Also: sin_sq, Gaussian, Lorentzian

6.4.3 square_wave

Usage:

#include <Level1/WindowFct.h>

row_vector square_wave(int size, int start, int finish);

Description:

The function ***square_wave*** returns a row vector containing a box function. The vector returned has the number of points specified by the argument ***size***. The box function will have value 1 between (and including) the points specified by the arguments ***start*** and "***finish*** and the value 0 elsewhere.

$$f_i = \begin{matrix} 1 \text{ when} & i \in [start, finish] \\ 0 \text{ when} & i \in [0, start), (finish, size-1] \end{matrix}$$

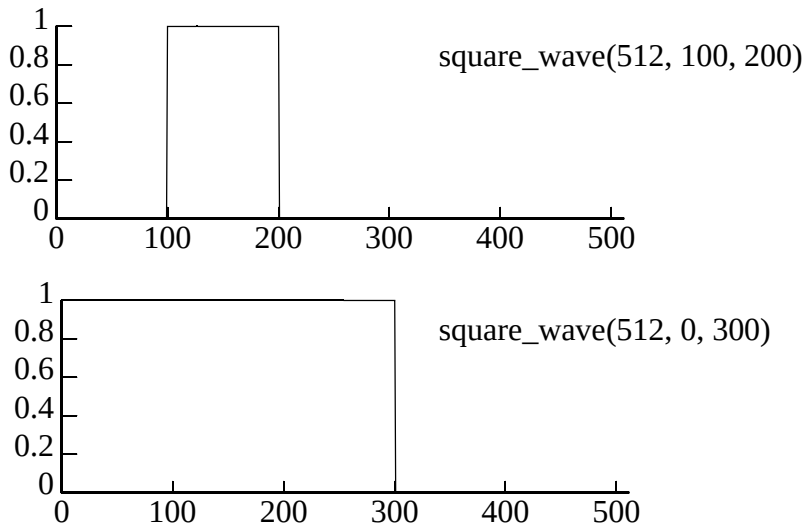
Return Value:

A row vector.

Example:

```
#include <gamma.h>
main()
  {
  row_vector BLK(512);
  BLK=square_wave(512,100,200);
  FM_1D("box1.mif",BLK,10,5,0,511);
  BLK=square_wave(512,0,300);
  FM_1D("box2.mif",BLK,10,5,0,511);
  }
```

In this example, the function square_wave is called twice with different parameters. Each is then written to a FrameMaker file. These have been imported directly into this document and shown in the following figures.



See Also: sin_sq, Gaussian, Lorentzian

6.4.4 Lorentzian

Usage:

```
#include <Level1/WindowFct.h>
row_vector Lorentzian(int size, int offset, int alpha);
```

Description:

This function is deprecated! More sophisticated Lorentzian functions are available in the chapter of that name in this document! Please use one supplied in that module rather than the one provided herein.

The window function ***Lorentzian*** returns a row vector containing a Lorentzian lineshape. The block returned has the size specified by the argument ***size***. The Lorentzian function will have a maximum value of 1 at the point specified by ***offset*** and the width is specified by the argument ***alpha***. The linewidth at half-height will be ***2*alpha***.

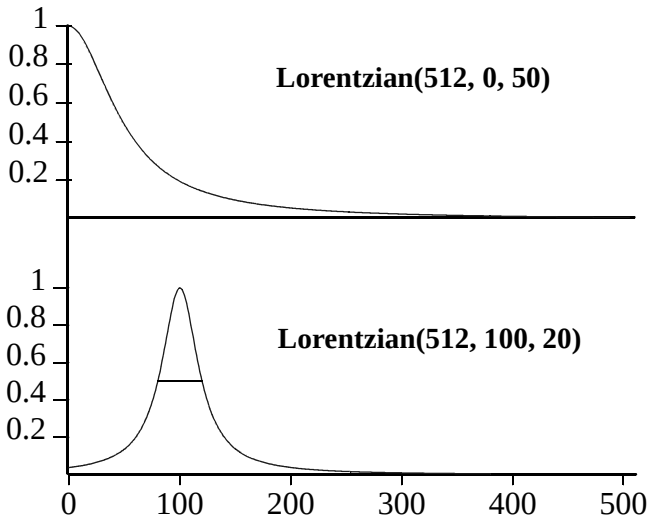
Return Value:

A row vector.

Example:

```
#include <gamma.h>
main()
  {
  row_vector BLK(512);
  BLK = Lorentzian(512, 0, 50);
  FM_1D ("Lorentz1.mif", BLK, 10, 5, 0, 511);
  BLK = Lorentzian(512, 100, 20);
  FM_1D ("Lorentz2.mif", BLK, 10, 5, 0, 511);
  }
```

In this example, the function Lorentzian is called twice with different parameters. Each is then written to a FrameMaker file. These have been imported directly into this document and shown in the following figures. Note that the linewidth at half-height in the second plot (marked) is 40, **twice the value input for alpha**.The default value for the offset is zero and the default value of alpha is 1.0.



See Also: sin_sq, Gaussian

6.4.5 Gaussian

Usage:

#include <Level1/WindowFct.h>

row_vector Gaussian(int size, int offset=0, double sigma=1);

Description:

The window function Gaussian returns a row vector containing a Gaussian lineshape. The block returned has the number of points specified by the argument “size”. The Gaussian function will have a maximum value of 1 at the point specified by “offset” and the width is specified by the argument “sigma”. The linewidth at half-height is given by formula ,

$$x_{1/2} = \sqrt{8ln(2)}\sigma = (2.35482)\sigma .$$

Thus, a sigma value of 42.46 produces a half-height linewidth of 100 points.

Note that Gaussian waveforms, in the context of shaped pulses for NMR simulations, are separately provided for in the pulse module. See that documentation book for use of Gaussian pulses.

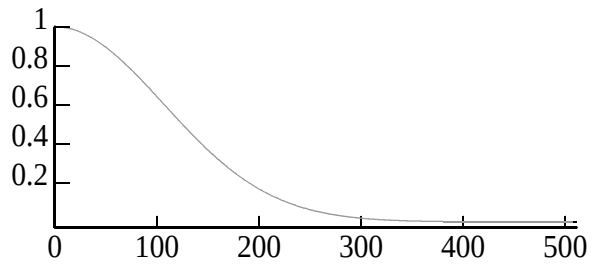
Return Value:

A row vector.

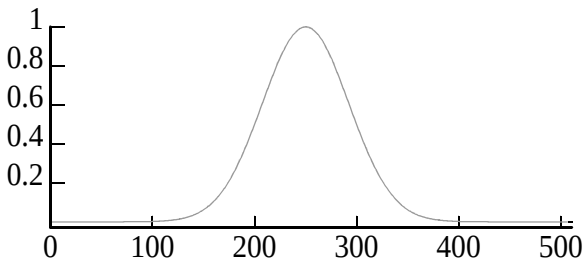
Example:

```
#include <gamma.h>
main()
  {
  row_vector BLK(512);
  BLK = Gaussian(512, 0, 106);
  FM_1D (“G1.mif”, BLK, 10, 5, 0, 511);
  BLK = Gaussian(512, 250, 42.46);
  FM_1D (“G2.mif”, BLK, 10, 5, 0, 511);
  }
```

In this example, the function ***Gaussian*** is called twice with different parameters. Each is then written to a FrameMaker file, G1.mif and G2.mif respectively. These have been imported directly into this document and shown in the following figures. Note that the linewidth at half-height increases with increasing ***sigma***. The default value for the offset is zero and the default value of ***sigma*** is 42.0.



Gaussian(512, 0, 106)



Gaussian(512, 250, 42.46)

See Also: sin_square, Lorentzian

6.4.6 sin_square

Usage:

#include <Level1/WindowFct.h>

row_vector sin_square(int size, int offset);

Description:

The window function ***sin_square*** returns a row vector containing a sin squared lineshape. The block returned has the size specified by the argument ***size***. The sin squared function will have a minimum of zero at the point specified by ***offset***. The function maximum is 1 and decreases to zero at the end of the row vector. The default value for the offset is zero.

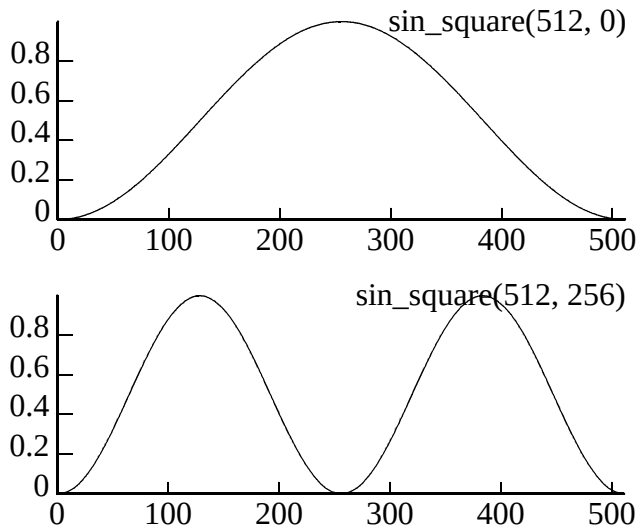
Return Value:

A 1D row vector.

Example:

```
#include <gamma.h>
main()
  {
  row_vector BLK(512);
  BLK = sin_square(512, 0);
  FM_1D ("ssq1.mif", BLK, 10, 5, 0, 511);
  BLK = sin_square(512, 256);
  FM_1D ("ssq2.mif", BLK, 10, 5, 0, 511);
  }
```

In this example, the function ***sin_square*** is called twice with different parameters. Each is then written to a FrameMaker file. These have been imported directly into this document and shown in the following figures.



See Also: Lorentzian, Gaussian

6.4.7 exponential

Usage:

#include <Level1/WindowFct.h>

row_vector exponential(int size, int offset=0, double alpha=0);

Description:

The window function exponential returns a row vector containing an exponential lineshape. The block returned has the number of points as specified by the argument ***size***. The exponential function will have value 1 at the point *offset* and is symmetric about this same point. The rate at which the exponential decays is dictated by the argument ***alpha***. There are four different ways which alpha is interpreted by the function.

1. alpha > 0: Alpha sets the linewidth (in points) at half-height, as is given by equation , by

$$x_{1/2} = \alpha[2ln2] = (1.3863)\alpha$$

2. alpha = [-1, 0): The | alpha| is the function intensity at points size+/-offset. For zero offset, alpha = -.3 produces a function starting at 1 exponentially decaying to 0.3.

3. alpha = 0: An appropriate alpha is chosen to decay the function to zero at points size+/-offset.

4. alpha < -1: The |alpha| is used and the function acts as if alpha > 0 was input.

Return Value:

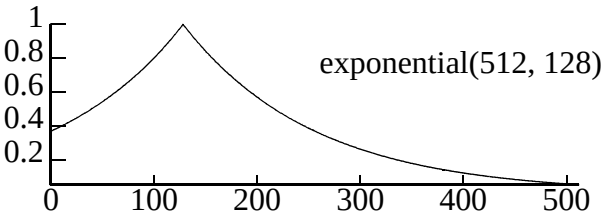
A row vector.

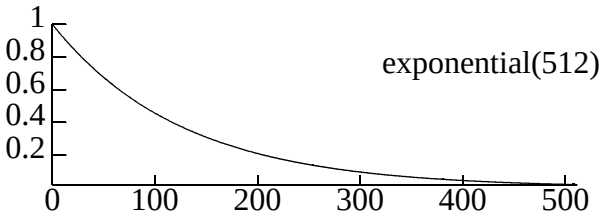
Examples:

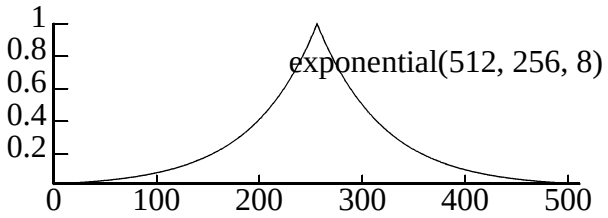
```
#include <gamma.h>
main()
  {
  row_vector BLK(512);
  BLK = exponential(512, 256, 8);
  FM_1D ("exp2.mif", BLK, 10, 5, 0, 511);
  }
```

In this example, the exponential function is written into row vector BLK and output to a FrameMaker file. The the plot was imported directly into this document and shown in the following figure along with other plots produced in the similar manner. (Note: the plots have been resized & labeled within FrameMaker).

See Also: sin_sq, Gaussian, Lorentzian







THESE NEED TO BE UPDATED !!!

6.4.8 Hanning

Usage:

```
#include <Level1/WindowFct.h>
row_vector Hanning(int size, int offset=0);
```

Description:

The window function Hanning returns a row vector containing an “Hanning” lineshape. The block returned has the number of points as specified by the argument “size”.

6.4.9 Hamming

Usage:

```
#include <Level1/WindowFct.h>
row_vector Hamming(int size, int offset=0);
```

Description:

The window function Hamming returns a row vector containing an “Hamming” lineshape. The block returned has the number of points as specified by the argument “size”.

6.4.10 hyperbol_sec

Usage:

#include <Level1/WindowFct.h>

row_vector hyperbol_sec(int size, int offset=0, double alpha = 38.0);

Description:

The window function ***hyperbol_sec*** returns a row vector containing an hyperbolic secant lineshape. The block returned has the number of points as specified by the argument ***size***. Furthermore, the function is centered about the offset point ***offset***. An additional parameter, ***alpha***, sets the linewidth at half height for the function in accordance with equation

$$x_{1/2} = \alpha[2\,\mathrm{acosh}(2)] = (2.64)\alpha$$

Thus, a sigma value of 37.88 (the approximate default value) produces a half-height linewidth of 100 points.

Return Value:

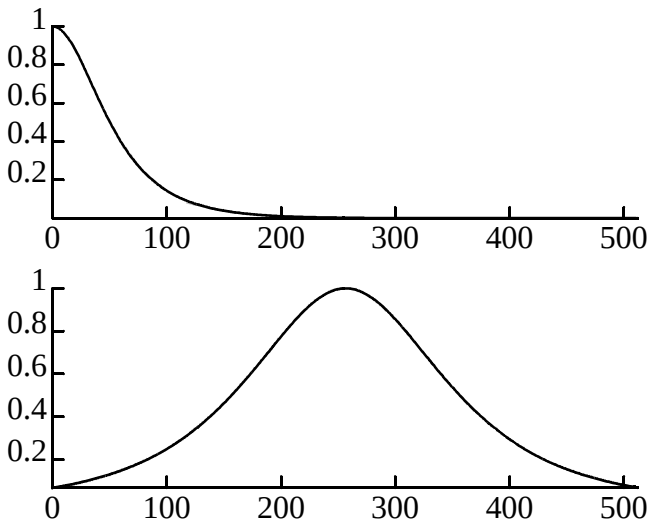
A 1D row vector.

Example:

```
#include <gamma.h>
main()
  {
  row_vector waveform;
   waveform = hyperbol_sec(512, 0);
   FM_1D("waveform1.mif", waveform, 13, 5, 0, 512);
   waveform = hyperbol_sec(512, 256, 75.76);
   FM_1D("waveform2.mif", waveform, 13, 5, 0, 512);
  }
```

In this example, the function hyperbol_sec is called twice with different parameters. Each is then written to a

FrameMaker file. These have been imported directly into this document and shown in the following figures.



6.4.11 Kaiser

Usage:

```
#include <Level1/WindowFct.h>
row_vector Hanning(int size, int offset=0);
```

Description:

The window function Kaiser returns a row vector containing an “Kaiser” lineshape. The block returned has the number of points as specified by the argument “size”.

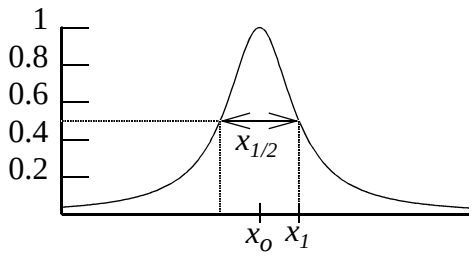
6.5 Description

6.5.1 Lorentzians

The Lorentzian function is given by the formula

$$f(x) = \frac{1}{\alpha^2 + (x - x_o)^2} \tag{6-1}$$

where the function maximizes to 1 at $x = x_o$ and is zero at $x = \pm\infty$. The value of α dictates the width of the lineshape, usually expressed in terms of the linewidth at half height, $x_{1/2}$.



The relationship between α and $x_{1/2}$ is derived as follows.

$$\frac{f(x_1)}{f(x_o)} = \frac{0.5}{1} = \frac{\alpha^2 + (x_o - x_o)^2}{\alpha^2 + (x_1 - x_o)^2}$$

$$\alpha^2 = 0.5[\alpha^2 + (x_1 - x_o)^2] = 0.5[\alpha^2 + (0.5x_{1/2})^2]$$

$$2\alpha = x_{1/2} \tag{6-2}$$

For the discrete, rather than continuous function, the function is

$$f_i = \frac{1}{\alpha^2 + (x_i - x_o)^2} \tag{6-3}$$

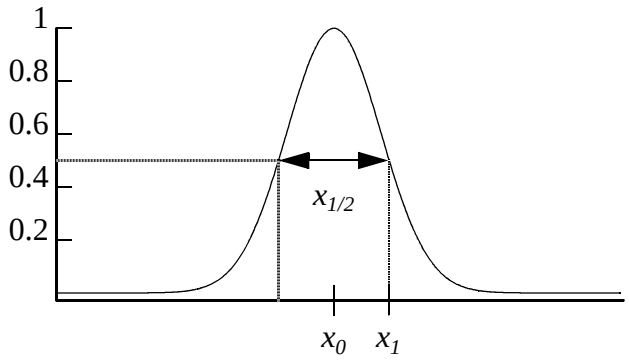
6.5.2 Gaussian

The Gaussian function is given by the formula

$$f(x) = exp\left[\frac{-(x - x_o)^2}{2\sigma^2}\right] \tag{6-4}$$

where the function maximizes to *1* at x = xo$x = x_o$ and is zero at $x = \pm\infty$. The value of σ dictates the width of the lineshape. This can be related to the linewidth at half height, $x_{1/2}$.

The relationship between σ and $x_{1/2}$ is derived as follows.



$$\frac{f(x_1)}{f(x_o)} = \frac{0.5}{1} = \frac{exp[-(x_1-x_0)^2/(2\sigma^2)]}{exp[-(x_0-x_0)^2/(2\sigma^2)]}$$

$$\frac{1}{2} = exp\left[\frac{-(x_1-x_0)^2}{2\sigma^2}\right] = exp\left[\frac{-(0.5x_{1/2})^2}{2\sigma^2}\right]$$

$$ln(0.5) = \frac{-(0.5x_{1/2})^2}{2\sigma^2} \qquad 2ln(2) = \frac{(0.5x_{1/2})^2}{\sigma^2} \qquad \frac{1}{2}x_{1/2} = \sigma\sqrt{2ln(2)}$$

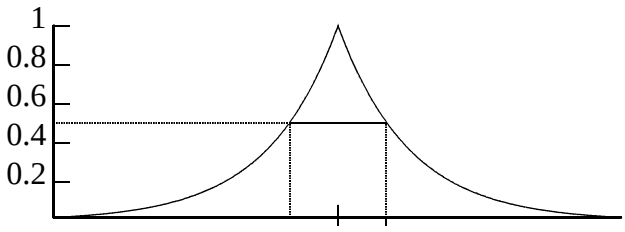
$$x_{1/2} = \sqrt{8ln(2)}\sigma = (2.35482)\sigma \tag{6-5}$$

6.5.3 Exponential

The exponential function is given by the formula

$$f(x) = exp\left(\frac{-|(x-x_o)|}{\alpha}\right) \tag{6-6}$$

where the function maximizes to 1 at $x = x_o$ and is zero at $x = \pm\infty$.



The relationship between α and $x_{1/2}$ is derived as follows.

$$\frac{f(x_1)}{f(x_o)} = \frac{0.5}{1} = \frac{exp[-|x_1 - x_0|/\alpha]}{exp[-|x_0 - x_0|/\alpha]} = exp[-|x_1 - x_0|/\alpha]$$

$$\frac{1}{2} = exp[-(0.5x_{1/2})/\alpha]$$

$$ln(0.5) = -(0.5x_{1/2})/\alpha \qquad 2ln(2) = x_{1/2}/\alpha \qquad x_{1/2} = \alpha[2ln2]$$

$$x_{1/2} = \alpha[2ln2] = (1.3863)\alpha \tag{6-7}$$

6.5.4 Hanning

The Hanning function is given by the formula[1]

$$f(x) = \frac{1}{2} + \frac{1}{2}\cos\left(\frac{\pi x}{2x_{max}}\right) \tag{6-8}$$

where x_{max} corresponds to the last point in the function. The function has a maximum value of 1 at $x = 0$ and a minimum of 1/2 at $x = x_{max}$

6.5.5 Hamming

The Hamming function is given by the formula[2]

$$f(x) = 0.54 + 0.46\cos\left(\frac{\pi x}{x_{max}}\right) \tag{6-9}$$

where x_{max} corresponds to the last point in the function. The function has a maximum value of 1 at $x = 0$ and a minimum of 0.54 at $x = x_{max}$

6.5.6 Hyperbolic Secant

The Hyperbolic secant function is given by the formula

$$f(x) = \text{sech}\left(\frac{x - x_0}{\alpha}\right) \tag{6-10}$$

The function has a maximum value of 1 at $x = x_0$ and has a width determined by the parameter α. The relationship between α and $x_{1/2}$ is derived as follows.

1. See [4], Ernst, R.R., Bodenhausen, G., and Wokaun, A., *Principles of Nuclear Magnetic Resonance in One and Two Dimensions*, page 103.
2. See [4], the same as in footnote 1 above.

$$\frac{f(x_1)}{f(x_o)} = \frac{0.5}{1} = \frac{\text{sech}\left(\frac{x_1 - x_0}{\alpha}\right)}{\text{sech}\left(\frac{x_0 - x_0}{\alpha}\right)} = \frac{\text{sech}[\alpha^{-1}(x_1 - x_0)]}{\text{sech}(0)} = \frac{1}{\cosh[x_{1/2}/(2\alpha)]}$$

$$\cosh[x_{1/2}/(2\alpha)] = 2 \qquad x_{1/2}/(2\alpha) = \text{acosh}(2)$$

$$x_{1/2} = \alpha[2\,\text{acosh}(2)] = (2.64)\alpha \tag{6-11}$$

6.5.7 Kaiser

The Kaiser function is given by the formula[1]

$$f(x) = \frac{J_0(\theta\sqrt{[1-(x/x_{max})^2]})}{J_0(\theta)} \tag{6-12}$$

where J_0 is the zero order modified Bessel function, x_{max} corresponds to the last point in the function and θ is an angle which must be specified. As θ increases, the Fourier transform of a function which has been multiplied by the Kaiser function will have an increased linewidth but a decreased "ripple" effect from truncation.

1. See [4], the same as in footnote 1 above.

7 Spherical Harmonics

7.1 Overview

The Spherical Hamonics module provides normalized spherical harmonics. Currently this includes the spherical harmonics through rank 4.

7.2 Available Functions

7.3 Discussion, Figures & Tables

7.4 Normalized Spherical Harmonics

7.4.1 Ylm, Ylmrad

Usage:

```
#include <Level1/SphHarmic.h>
complex Ylm(const int l, const int m, const double theta, const double phi);
complex Ylmrad(const int l, const int m, const double theta, const double phi);
```

Description:

Functions ***Ylm*** and ***Ylmrad*** return a complex number which is the value of the normalized spherical harmonic,

$$Y_{l,\,m}(\theta, \phi) \ ,$$

where θ and ϕ are the polar angles in the standard right handed spherical coordinate system (θ the angle down from the z-axis and ϕ the angle over from the x axis). For the two indices, ***l*** is the rank and ***m*** the angular momentum component. The angles are input in degrees for function ***Ylm*** and in radians for function ***Ylmrad***. Currently only the functions for ***l = 0, 1, 2, 3, & 4*** are provided. The higher rank spherical harmonics may be built up *via* spatial tensor cross products if desired, see class ***Space_T***.

Return Value:

A complex number which is the value of the normalized spherical harmonic for the angles specified.

Example:

```
#include <gamma.h>
main()
  {
  complex z;                          // Define a complex number.
  z = Ylm(2, -2, 0.0, 90.0);          // set z to Y2,-2(0, 90).
  }
```

Mathematical Basis:

Specific equations can be found in Figure B

See Also: Y00, Y10, Y11, Y1m1, Y20, Y21, Y2m1, Y22, Y2m2

7.4.2 Y00, Y00rad

Usage:

```
#include <Level1/SphHarmic.h>
double Y00();
```

Description:

The function ***Y00*** returns a double precision number which is the value of the normalized spherical harmonic,

$$Y_{0,0}(\theta, \phi) = \sqrt{\frac{1}{4\pi}}$$

as given in Figure B on page 120. As this is independent of the spherical angles θ and ϕ so they are not required as function arguments.

Return: A double precision number.

Example:

```
#include <gamma.h>
main()
  { double r= Y00(); }                    // Set r to Y0,0.
```

See Also: Ylm, Y10, Y11, Y1m1, Y20, Y21, Y2m1, Y22, Y2m2

0.0.1 Y10, Y10rad
7.4.3 Y11, Y11rad
7.4.4 Y1m1, Y1m1rad

Usage:

```
#include <Level1/SphHarmic.h>
double Y10 (const double theta);
complex Y11 (const double theta, constant double phi);
complex Y1m1 (const double theta, const double phi);
```

Description:

The functions ***Y11*** and***Y1m1*** return complex numbers which are values of the normalized spherical harmonics,

$$Y_{1,0}(\theta,\phi) = \sqrt{\frac{3}{4\pi}}\cos\theta \qquad Y_{1,1}(\theta,\phi) = -\sqrt{\frac{3}{8\pi}}\sin\theta e^{i\phi} \qquad Y_{1,-1}(\theta,\phi) = \sqrt{\frac{3}{8\pi}}\sin\theta e^{-i\phi}$$

where the angles ***theta*** and ***phi*** are input in degrees.

Return Value: A complex number.

Examples:

```
#include <gamma.h>
main()
  {
  complex z;                        // define a complex number.
  double r = Y10(45.0);             // declare a double, set to Y1,0(PI/4).
  z = Y11(90., 90.);                // set z to Y1,1(PI/2, PI/2).
  z = Y1m1(90., 90.);               // set z to Y1,-1(PI/2, PI/2).
  }
```

See Also: Ylm, Y00, Y10, Y1m1, Y20, Y21, Y2m1, Y22, Y2m2

0.0.2 Y20 Y20rad
7.4.5 Y21, Y21rad
7.4.6 Y2m1, Y2m1rad
7.4.7 Y22, Y22rad
7.4.8 Y2m2, Y2m2rad

Usage:

```
#include <Level1/SphHarmic.h>
    double Y20 (const double theta);
    complex Y21 (const double theta, constant double phi);
    complex Y2m1 (const double theta, const double phi);
complex Y22 (const double theta, constant double phi);
complex Y2m2 (const double theta, const double phi);
```

Description:

Functions ***Y2x*** and ***Y2mx*** return either a double or complex value of the normalized spherical harmonic,

$$Y_{2,0}(\theta,\phi) = \sqrt{\frac{5}{16\pi}}(3\cos^2\theta - 1)$$

$$Y_{2,1}(\theta,\phi) = -\sqrt{\frac{15}{8\pi}}\cos\theta\sin\theta e^{i\phi} \qquad Y_{2,-1}(\theta,\phi) = \sqrt{\frac{15}{8\pi}}\cos\theta\sin\theta e^{-i\phi}$$

$$Y_{2,2}(\theta,\phi) = \sqrt{\frac{15}{32\pi}}\sin^2\theta e^{2i\phi} \qquad Y_{2,-2}(\theta,\phi) = \sqrt{\frac{15}{32\pi}}\sin^2\theta e^{-2i\phi}$$

where *x* spans [0, 2]. The angles ***theta*** and ***phi*** are input in degrees.

Return Value: A double precision or complex number.

Examples:

```
#include <gamma.h>
main()
  {
  double r;                          // define a double precision number.
  r = Y20(45.0);                     // set r to Y2,0(PI/4).
  complex z;
  z = Y21(90., 90.);                 // set z to Y2,1(PI/2, PI/2).
  z = Y2m1(90., 90.);                // set z to Y2,-1(PI/2, PI/2).
  z = Y22(90., 90.);                 // set z to Y2,2(PI/2, PI/2).
  z = Y2m2(90., 90.);                // set z to Y2,-2(PI/2, PI/2).
  }
```

See Also: Ylm, Y00, Y10, Y11, Y1m1, Y21, Y2m1, Y22, Y2m2

0.0.3 Y30 Y30rad
7.4.9 Y31, Y31rad
7.4.10 Y3m1, Y3m1rad
7.4.11 Y32, Y32rad
7.4.12 Y3m2, Y3m2rad
7.4.13 Y33, Y33rad
7.4.14 Y3m3, Y3m3rad

Usage:

```
#include <Level1/SphHarmic.h>
    double Y30 (const double theta);
    double Y30rad (const double theta);
    complex Y31(const double theta, constant double phi);
    complex Y3m1 (const double theta, const double phi);
    complex Y32(const double theta, constant double phi);
    complex Y3m2(const double theta, constant double phi);
    complex Y33 (const double theta, constant double phi);
    complex Y3m3 (const double theta, const double phi);
```

Description:

Functions ***Y3x*** and ***Y3mx*** return real (x=0) or complex values of a rank 3 normalized spherical harmonic,

$$Y_{3,0}(\theta,\phi) = \sqrt{\frac{7}{16\pi}}(2\cos^3\theta - 3\cos\theta\sin^2\theta) \qquad Y_{3,\pm2}(\theta,\phi) = \sqrt{\frac{105}{32\pi}}\cos\theta\sin^2\theta e^{(\pm2)i\phi}$$

$$Y_{3,\pm1}(\theta,\phi) = \mp\sqrt{\frac{21}{64\pi}}(4\cos^2\theta\sin\theta - \sin^3\theta)e^{\pm i\phi} \qquad Y_{3,\pm3}(\theta,\phi) = \mp\sqrt{\frac{35}{64\pi}}\sin^3\theta e^{(\pm3)i\phi}$$

where *x* spans [0, 3]. The angles theta and phi are input in degrees. Function whos names end with rad are equivalent to those without the suffix but take angles in radians rather than degrees.

Return Value: A complex number or a real number.

Mathematical Basis: See function Ylm.

Examples:

```
#include <gamma.h>
main()
  {
  complex z;                        // define a complex number.
  r = Y30(45.0);                    // set r to Y3,0(PI/4).
  z = Y31(90., 90.);                // set z to Y3,1(PI/2, PI/2).
  z = Y3m1(90., 90.);               // set z to Y3,-1(PI/2, PI/2).
  z = Y32(90., 90.);                // set z to Y32,2(PI/2, PI/2).
  z = Y3m2(90., 90.);               // set z to Y3,-2(PI/2, PI/2).
  z = Y33(90., 90.);                // set z to Y3,3(PI/2, PI/2).
  z = Y3m3(90., 90.);               // set z to Y3,-3(PI/2, PI/2).
  }
```

See Also: Ylm, Y00, Y10, Y11, Y1m1, Y20, Y21, Y2m1, Y2m2

7.5 Description

The spherical harmonics are specified mathematically as

$$Y_{l,\,m}(\theta,\,\phi) \tag{7-1}$$

where l is the rank, m the component, and θ, ϕ the standard spherical angles.

GAMMA Spherical Coordinates

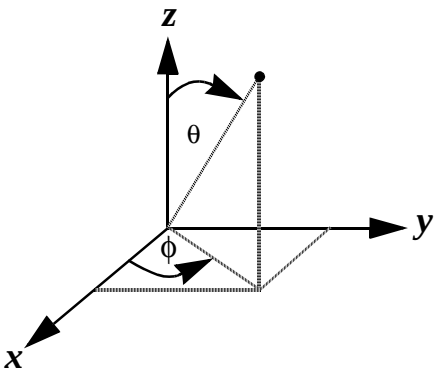


Figure A GAMMA defined spherical angles. Angle ϕ spans [0, 2π] where 0 coincides with the +x axis & $\pi/2$ with the +y axis. Angle θ spans [0, π] where 0 coincides with the +z axis and $\pi/2$ with the xy plane.

The exact definition of the spherical harmonics will vary in the literature. By and large, such variations are the result of differences in the definition of coordinate axes and spherical angles. Often, formulae for the spherical harmonics obtained from different sources can be made to coincide by replacement of the angle ϕ with $-\phi$ (on one side of all algebraic formulas). We use the more common definition, maintaining the relationship

$$[Y_{l,\,m}(\theta,\,\phi)]^{*} \;=\; (-1)^{m} Y_{l,\,-m}(\theta,\,\phi)\,. \tag{7-2}$$

Another useful formula involving spherical harmonics is the orthogonality relationship

$$\delta_{l,\,l'}\delta_{m,\,m'} \;=\; \int_{\phi=0}^{2\pi}\int_{\theta=0}^{\pi} [Y_{l,\,m}(\theta,\,\phi)]^{*} Y_{l',\,m'}(\theta,\,\phi)\sin\theta\, d\theta\, d\varphi \tag{7-3}$$

GAMMA's normalized spherical harmonics are listed through rank 4 in the following figure[1]

1. The spherical harmonics through rank 3 can be found in Edmonds, Table1, page 124. A second source of these functions, through rank 4, is Weissbluth, TABLE 1.1, p 4.

Normalized Spherical Harmonics - Ranks 0 Through 4

$$Y_{0,0}(\theta,\phi) = \sqrt{\frac{1}{4\pi}} \qquad Y_{1,0}(\theta,\phi) = \sqrt{\frac{3}{4\pi}}\cos\theta \qquad Y_{1,\pm1}(\theta,\phi) = \mp\sqrt{\frac{3}{8\pi}}\sin\theta e^{\pm i\phi}$$

$$Y_{2,0}(\theta,\phi) = \sqrt{\frac{5}{16\pi}}(3\cos^2\theta - 1) \qquad Y_{2,\pm1}(\theta,\phi) = \mp\sqrt{\frac{15}{8\pi}}\cos\theta\sin\theta e^{\pm i\phi}$$

$$Y_{2,\pm2}(\theta,\phi) = \sqrt{\frac{15}{32\pi}}\sin^2\theta e^{\pm 2i\phi}$$

$$Y_{3,0}(\theta,\phi) = \sqrt{\frac{7}{16\pi}}(2\cos^3\theta - 3\cos\theta\sin^2\theta)$$

$$Y_{3,\pm1}(\theta,\phi) = \mp\sqrt{\frac{21}{64\pi}}(4\cos^2\theta\sin\theta - \sin^3\theta)e^{\pm i\phi}$$

$$Y_{3,\pm2}(\theta,\phi) = \sqrt{\frac{105}{32\pi}}\cos\theta\sin^2\theta e^{\pm 2i\phi} \qquad Y_{3,\pm3}(\theta,\phi) = \mp\sqrt{\frac{35}{64\pi}}\sin^3\theta e^{\pm 3i\phi}$$

$$Y_{4,0}(\theta,\phi) = \sqrt{\frac{9}{256\pi}}(35\cos^4\theta - 30\cos^2\theta + 3)$$

$$Y_{4,\pm1}(\theta,\phi) = \mp\sqrt{\frac{45}{64\pi}}\sin\theta(7\cos^3\theta - 3\cos\theta)e^{\pm i\phi}$$

$$Y_{4,\pm2}(\theta,\phi) = \sqrt{\frac{45}{128\pi}}\sin^2\theta(7\cos^2\theta - 1)e^{\pm 2i\phi}$$

$$Y_{4,\pm3}(\theta,\phi) = \mp\sqrt{\frac{315}{64\pi}}\sin^3\theta\cos\theta e^{\pm 3i\phi} \qquad Y_{4,\pm4}(\theta,\phi) = \mp\sqrt{\frac{315}{512\pi}}\sin^4\theta e^{\pm 4i\phi}$$

Figure B Normalized spherical harmonics used in GAMMA.

Two commutation relationships involving spherical harmonics may also be applied. These are set in terms of angular momentum,

$$[\boldsymbol{J}_z, Y_{l,m}(\theta,\phi)] = -i\frac{\partial}{\partial\phi}Y_{l,m}(\theta,\phi) = mY_{l,m}(\theta,\phi) \tag{7-4}$$

and

$$[\boldsymbol{J}_\pm, Y_{l,m}(\theta,\phi)] = -ie^{\pm i\phi}\left[-\cot\beta\frac{\partial}{\partial\phi} \pm i\frac{\partial}{\partial\theta}\right]Y_{l,m} = \sqrt{l(l+1) - m(m\pm1)}\,Y_{l,m} \tag{7-5}$$

using spatial forms of angular momentum operators[1].

Selective Values of Normalized Spherical Harmonics

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

θ ϕ	00	11	10	22	21	20	33	32	31	30
0 0	0.282	0.0	0.489	0.0	0.0	0.631	0.0	0.0	0.0	0.746
30	0.282	0.0	0.489	0.0	0.0	0.631	0.0	0.0	0.0	0.746
45	0.282	0.0	0.489	0.0	0.0	0.631	0.0	0.0	0.0	0.746
60	0.282	0.0	0.489	0.0	0.0	0.631	0.0	0.0	0.0	0.746
90	0.282	0.0	0.489	0.0	0.0	0.631	0.0	0.0	0.0	0.746
180	0.282	0.0	0.489	0.0	0.0	0.631	0.0	0.0	0.0	0.746
30 0	0.282	-0.173	0.423	0.097	-0.335	0.394	-0.052	0.221	-0.444	0.242
30	0.282	-0.150-0.086	0.423	0.048+0.084i	-0.290-0.167i	0.394	-0.052i	0.111+0.192i	-0.385-0.222i	0.242
45	0.282	-0.122-0.122i	0.423	0.097i	-0.237-0.237i	0.394	0.037-0.037i	0.221i	-0.314-0.314i	0.242
60	0.282	-0.086-0.150i	0.423	-0.048+0.084i	-0.167-0.290i	0.394	0.052	-0.111+0.192i	-0.222-0.385i	0.242
90	0.282	-0.173i	0.423	-0.097	-0.335i	0.394	0.052i	-0.221	-0.444i	0.242
180	0.282	0.173	0.423	0.097	0.335	0.394	0.052	0.221	0.444	0.242
45 0	0.282	-0.244	0.345	0.193	-0.386	0.158	-0.148	0.361	-0.343	-0.132
30	0.282	-0.212-0.122i	0.345	0.097+0.167i	-0.335-0.193i	0.158	-0.148i	0.181+0.313i	-0.297-0.171i	-0.132
45	0.282	-0.173-0.173i	0.345	0.193i	-0.273-0.273i	0.158	0.104-0.104i	0.361i	-0.242-0.242i	-0.132
60	0.282	-0.122-0.212i	0.345	-0.097+0.167i	-0.193-0.335i	0.158	0.148	-0.181+0.313i	-0.171-0.297i	-0.132
90	0.282	-0.244i	0.345	-0.193	-0.386i	0.158	0.148i	-0.361	-0.343i	-0.132
180	0.282	0.244	0.345	0.193	0.386	0.158	0.148	0.361	0.343	-0.132
60 0	0.282	-0.299	0.244	0.290	-0.335	-0.079	-0.271	0.383	0.070	-0.326
30	0.282	-0.259-0.150i	0.244	0.145+0.251i	-0.290-0.167i	-0.079	-0.271i	0.192+0.332i	-0.061-0.035i	-0.326
45	0.282	-0.212-0.212i	0.244	0.290i	-0.237-0.237i	-0.079	0.192-0.192i	0.383i	-0.050-0.050i	-0.326
60	0.282	-0.150-0.259i	0.244	-0.145+0.251i	-0.167-0.290i	-0.079	0.271	-0.192+0.332i	-0.035-0.061i	-0.326
90	0.282	-0.299i	0.244	-0.290	-0.335i	-0.079	0.271i	-0.383	-0.070i	-0.326
180	0.282	0.299	0.244	0.290	0.335	-0.079	0.271	0.383	0.070	-0.326
90 0	0.282	-0.345	0.0	0.386	0.0	-0.315	-0.417	0.0	0.323	0.0
30	0.282	-0.299-0.173i	0.0	0.193+0.335i	0.0	-0.315	-0.417i	0.0	0.280+0.162i	0.0
45	0.282	-0.244-0.244i	0.0	0.386i	0.0	-0.315	0.295-0.295i	0.0	0.229+0.229i	0.0
60	0.282	-0.173-0.299i	0.0	-0.193+0.335i	0.0	-0.315	0.417	0.0	0.162+0.280i	0.0
90	0.282	-0.346i	0.0	-0.386	0.0	-0.315	0.417i	0.0	0.323i	0.0
180	0.282	0.345	0.0	0.386	0.0	-0.315	0.417	0.0	-0.323	0.0

Figure C : Selected values for the normalized spherical Harmonics provided by the function Ylm. The values for m being a negative integer are obtained from the relationship in equation (7-2)[1].

1. The spatial forms of the operators $\boldsymbol{J_z}$ and $\boldsymbol{J_\pm}$ can be found in Brink and Satchler, page 18, equations (2.5) and (2.6) respectively. The commutation relationships are equivalent to those applicable to tensors, see equation (4.9) on page 53 of Brink and Satchler. The spherical harmonics, $\boldsymbol{Y_{l,m}}$, in this latter context are the tensor components $\boldsymbol{T_{l,m}}$.

1. These were computed independently of GAMMA and useful as a check of function performance.

8 Wigner Rotations

8.1 Overview

The Wigner rotations module contains various functions of spatial coordinates which are not associated to any specific class in GAMMA. Currently this includes the Wigner rotation matrix elements for any rank.

8.2 Available Functions

Reduced Wigner Rotation Matrix Elements

8.3 Discussion Sections

8.4 Figures

8.5 Reduced Elements

8.5.1 d0

Usage:

```
#include <spatial.h>
double d0();
```

Description:

The function *d0* returns a double containing the value of the rank 0 reduced Wigner rotation matrix element

$$d^{(0)}_{0,0}(\beta) \equiv 1 \ . \tag{8-1}$$

This function is provided for consistency with the other reduced Wigner rotation matrix element functions included in the spatial functions library.

Return :

A double precision number.

Example(s):

```
#include <gamma.h>
main()
  { double r = d0(); }                    // Set r to d0,0.
```

Mathematical Basis:

There is only one reduced (rank 0) Wigner rotation matrix element. This is identically unity for all angles[1].

See Also: d1half, d1, d2, dJ

8.5.2 d1half

Usage:

```
double d1half(const int m, const int n, const double beta);
```

Description:

The function ***d1half*** returns a double value of the rank 1/2 reduced Wigner rotation matrix element

$$d^{(1/2)}_{m,n}(\beta) \ ,$$

where β is supplied in degrees and the integers ***m*** and ***n*** supplied in **units of 1/2**.

Return Value:

A double precision number.

Example:

```
#include <gamma.h>
```

1. See Brink and Satchler, page 24, TABLE 1.

```
main()
  { double r = d1half(-1,-1,90.0); }          // Set r to d-1/2,-1/2(PI/2).
```

The reduced (rank 1) Wigner rotation matrix elements supplied by this function are given in the following figure[1].

Reduced Rank 1/2 Wigner Rotation Matrix Elements

$$d^{(1/2)}_{m,n}(\beta)$$

m \ n	**1/2**	**-1/2**
1/2	$\cos(\beta/2)$	$-\sin(\beta/2)$
-1/2	$\sin(\beta/2)$	$\cos(\beta/2)$

Table 19-1 : The reduced Wigner rotation matrix elements of rank 1/2. The angle β is the standard Euler angle.

The full Wigner rotation matrix elements are related to these *via*

$$\boldsymbol{D}^{(1/2)}_{m,n}(\alpha,\beta,\gamma) = e^{-i\alpha m} d^{(1/2)}_{m,n}(\beta) e^{-i\gamma n} \quad . \tag{8-2}$$

Other useful relationships concerning these elements are

$$d^{(1/2)}_{m,n}(\beta) = (-1)^{m-n} d^{(1/2)}_{n,m}(\beta) = (-1)^{m-n} d^{(1/2)}_{-n,-m}(\beta) \quad . \tag{8-3}$$

Examples of the $d^{(1/2)}_{m,n}(\beta)$ matrix are the following.

$$d^{(1/2)}_{m,n}\left(\frac{\pi}{3}\right) = \begin{bmatrix} 0.866 & -0.5 \\ 0.5 & 0.866 \end{bmatrix} \qquad d^{(1/2)}_{m,n}\left(\frac{\pi}{2}\right) = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 & -1 \\ 1 & 1 \end{bmatrix} \qquad d^{(1/2)}_{m,n}(\pi) = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}$$

Note that the above arrays correspond to $\boldsymbol{R}^{(1/2)}_{y}(\beta)$ and have the basis function ordering of that presented in the previous table.

See Also: d0, d1, d2, dJ

1. See Brink and Satchler, page 24, TABLE 1.

8.5.3 d1

Usage:

```
#include <spatial.h>
double d1(const int m, const int n, const double beta);
```

Description:

The function ***d1*** returns a double precision number containing the value of the rank 1 reduced Wigner rotation matrix element

$$d^{(1)}_{m,n}(\beta) ,$$

where β is the rotation angle input in degrees. The angular momentum indices ***m*** and ***n*** are input as integers an are both in the range [-1, 1].

Return Value:

A double precision number.

Example:

```
#include <gamma.h>
main()
  { double r = d1(1,-1,90.0); }                    // set r to d1(1,-1, PI/2).
```

Mathematical Basis:

The reduced (rank 1) Wigner rotation matrix elements supplied by this function are given in the following figure[1].

Reduced Rank 1 Wigner Rotation Matrix Elements $d^{(1)}_{m,n}(\beta)$

$d^{(1)}_{m,n}(\beta)$	$n = 1$	$n = 0$	$n = -1$
$m = 1$	$\cos^2(\beta/2)$	$-\sqrt{1/2}\sin\beta$	$\sin^2(\beta/2)$
$m = 0$	$\sqrt{1/2}\sin\beta$	$\cos\beta$	$-\sqrt{1/2}\sin\beta$
$m = -1$	$\sin^2(\beta/2)$	$\sqrt{1/2}\sin\beta$	$\cos^2(\beta/2)$

Figure 8-A The reduced Wigner rotation matrix elements of rank 1. The angle β is the standard Euler angle.

The full Wigner rotation matrix elements are related to these *via*

$$\boldsymbol{D}^{(1)}_{m,n}(\alpha, \beta, \gamma) = e^{-i\alpha m} d^{(1)}_{m,n}(\beta) e^{-i\gamma n} . \tag{8-4}$$

Other useful relationships concerning these elements are

$$d^{(1)}_{m,n}(\beta) = (-1)^{m-n} d^{(1)}_{n,m}(\beta) = (-1)^{m-n} d^{(1)}_{-n,-m}(\beta) \quad . \tag{8-5}$$

From (8-5) one may surmise that there are actually only four unique elements as shown in the following figure[2].

1. See Brink and Satchler, page 24, TABLE 1.
2. See Brink and Satchler, page 24, TABLE 1.

Reduced Rank 1 Wigner Rotation Matrix Elements $d^{(1)}_{m,n}(\beta)$

$d^{(1)}_{m,n}(\beta)$	$n = 1$	$n = 0$	$n = -1$
$m = 1$	$\cos^2(\beta/2)$	$-\sqrt{1/2}\sin\beta$	$\sin^2(\beta/2)$
$m = 0$	$-d^{(1)}_{1,0}(\beta)$	$\cos\beta$	$d^{(1)}_{1,0}(\beta)$
$m = -1$	$d^{(1)}_{1,-1}(\beta)$	$-d^{(1)}_{1,0}(\beta)$	$d^{(1)}_{1,1}(\beta)$

Figure 8-B The reduced Wigner rotation matrix elements of rank 1 explicitly displaying the symmetry between the elements. The angle β is the standard Euler angle.

Examples of the $d^{(1)}_{m,n}(\beta)$ matrix are the following.

$$d^{(1)}_{m,n}\left(\frac{\pi}{3}\right) = \frac{1}{4}\begin{bmatrix} 3 & -2.565 & 1 \\ 2.565 & 2 & -2.565 \\ 1 & 2.565 & 3 \end{bmatrix} \quad d^{(1)}_{m,n}\left(\frac{\pi}{2}\right) = \frac{1}{2}\begin{bmatrix} 1 & -\sqrt{2} & 1 \\ \sqrt{2} & 0 & -\sqrt{2} \\ 1 & \sqrt{2} & 1 \end{bmatrix} \quad d^{(1)}_{m,n}(\pi) = \begin{bmatrix} 0 & 0 & 1 \\ 0 & -1 & 0 \\ 1 & 0 & 0 \end{bmatrix}$$

See Also:

8.5.4 d2

Usage:

```
#include <spatial.h>
double d2(const int m, const int n, const double beta);
```

Description:

The function d2 returns a double precision number containing the value of the rank 2 reduced Wigner rotation matrix element

$$d^{(2)}_{m,n}(\beta) ,$$

where β is supplied in degrees and the indices m and n supplied as integers.

Return Value:

A double precision number.

Example:

```
#include <gamma.h>
main()
  { double r = d2(2,-2,90.0); }                 // set r to d2,2(PI/2).
```

Mathematical Basis:

The reduced (rank 2) Wigner rotation matrix elements supplied by this function are given in the following figure[1].

1. See Brink and Satchler, page 24, TABLE 1.

Reduced Rank 2 Wigner Rotation Matrix Elements

$$d^{(2)}_{m,n}(\beta)$$

m \ n	**2**	**1**	**0**	**-1**	**-2**
2	$\cos^4(\beta/2)$	$-\frac{1}{2}(1+\cos\beta)\sin\beta$	$\sqrt{3/8}\sin^2\beta$	$\frac{1}{2}(\cos\beta-1)\sin\beta$	$\sin^4(\beta/2)$
1	$\frac{1}{2}\sin\beta(\cos\beta+1)$	$\cos^2\beta-\frac{1}{2}(1-\cos\beta)$	$-\sqrt{3/2}\sin\beta\cos\beta$	$\frac{1}{2}(1+\cos\beta)-\cos^2\beta$	$\frac{1}{2}\sin\beta(\cos\beta-1)$
0	$\sqrt{3/8}\sin^2\beta$	$\sqrt{3/8}\sin(2\beta)$	$\frac{1}{2}(3\cos^2\beta-1)$	$-\sqrt{3/8}\sin(2\beta)$	$\sqrt{3/8}\sin^2\beta$
-1	$-\frac{1}{2}(\cos\beta-1)\sin\beta$	$\frac{1}{2}(1+\cos\beta)-\cos^2\beta$	$\sqrt{3/2}\sin\beta\cos\beta$	$\cos^2\beta-\frac{1}{2}(1-\cos\beta)$	$-\frac{1}{2}(1+\cos\beta)\sin\beta$
-2	$\sin^4(\beta/2)$	$-\frac{1}{2}(\cos\beta-1)\sin\beta$	$\sqrt{3/8}\sin^2\beta$	$\frac{1}{2}(1+\cos\beta)\sin\beta$	$\cos^4(\beta/2)$

Table 19-2 The reduced Wigner rotation matrix elements of rank 2. The angle β is the standard Euler angle

The full Wigner rotation matrix elements are related to these *via*

$$\boldsymbol{D}^{(2)}_{m,n}(\alpha,\beta,\gamma) = e^{-i\alpha m} d^{(2)}_{m,n}(\beta) e^{-i\gamma n} \quad . \tag{8-6}$$

Other useful relationships concerning these elements are

$$d^{(2)}_{m,n}(\beta) = (-1)^{m-n} d^{(2)}_{n,m}(\beta) = (-1)^{m-n} d^{(2)}_{-n,-m}(\beta) \quad . \tag{8-7}$$

This equation simplifies the above table of 25 elements down to a table containing only 9 unique components.

Reduced Rank 2 Wigner Rotation Matrix Elements

$$d^{(2)}_{m,n}(\beta)$$

m \ n	**2**	**1**	**0**	**-1**	**-2**
2	$\cos^4(\beta/2)$	$-\frac{1}{2}(1+\cos\beta)\sin\beta$	$\sqrt{3/8}\sin^2\beta$	$\frac{1}{2}(\cos\beta-1)\sin\beta$	$\sin^4(\beta/2)$
1	$-d^{(2)}_{2,1}(\beta)$	$\left(\cos\beta-\frac{1}{2}\right)(\cos\beta+1)$	$-\sqrt{3/2}\sin\beta\cos\beta$	$\left(\cos\beta+\frac{1}{2}\right)(1-\cos\beta)$	$d^{(2)}_{2,-1}(\beta)$
0	$d^{(2)}_{2,0}(\beta)$	$-d^{(2)}_{1,0}(\beta)$	$\frac{1}{2}(3\cos^2\beta-1)$	$d^{(2)}_{1,0}(\beta)$	$d^{(2)}_{2,0}(\beta)$
-1	$-d^{(2)}_{2,-1}(\beta)$	$d^{(2)}_{1,-1}(\beta)$	$-d^{(2)}_{1,0}(\beta)$	$d^{(2)}_{1,1}(\beta)$	$d^{(2)}_{2,1}(\beta)$
-2	$d^{(2)}_{2,-2}(\beta)$	$-d^{(2)}_{2,-1}(\beta)$	$d^{(2)}_{2,0}(\beta)$	$-d^{(2)}_{2,1}(\beta)$	$d^{(2)}_{2,2}(\beta)$

Table 19-3 The reduced Wigner rotation matrix elements of rank 2 explicitly displaying the symmetry between the elements. There are 9 unique elements. The angle β is the standard Euler angle.

Examples of the $d^{(2)}(\beta)$ matrix are the following.

$$d^{(2)}\left(\frac{\pi}{2}\right) = \frac{1}{2}\begin{bmatrix} \frac{1}{2} & -1 & \sqrt{3/2} & -1 & \frac{1}{2} \\ 1 & -1 & 0 & 1 & -1 \\ \sqrt{3/2} & 0 & -1 & 0 & \sqrt{3/2} \\ 1 & 1 & 0 & -1 & -1 \\ \frac{1}{2} & 1 & \sqrt{3/2} & 1 & \frac{1}{2} \end{bmatrix} \qquad d^{(2)}(\pi) = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{bmatrix}$$

See Also:

8.5.5 dJ

Usage:

```
#include <spatial.h>
double dJ(const int J, const int m, const int n, const double beta);
matrix dJ(const int J, const double beta);
```

Description:

The function dJ returns values for the rank J reduced Wigner rotation matrix or its elements.

1. dJ(int J, int m, int n, double beta) - Returns the m, n element of the rank J reduced Wigner rotation matrix for the Euler angle beta.

$$d^{(J)}_{m,n}(\beta) .$$

2. dJ(int J, double beta) - Returns the rank J reduced Wigner rotation matrix for the Euler angle beta.

$$d^{(J)}(\alpha, \beta, \gamma)$$

The Euler angle β is input in degrees. The angular momentum indices m and n are taken as integers. If J is given as a negative number, it is assumed that the user desires half integral J. For example, if J = -3 the function evaluated is

$$d^{(5/2)}_{m,n}(\beta) ,$$

and m,n = -3 corresponds to -5/2. The full correspondence between the input integer for J in the function argument and the J evaluated is given in the following table.

Mapping of J Input versus J Evaluated for Function dJ

Figure1: Reduced Wigner Rotation Matrix Elements

J		m, n	
input	evaluated	input	evaluated
0	0	0	0
1	1	1	1
		0	0
		-1	-1
2	2	2	2
		1	1
		0	0
		-1	-1
		-2	-2
3	3	3	3
		2	2
		1	1
		0	0
		-1	-1
		-2	-2
		-3	-3
J	J	J	J
		⋮	⋮
		-j	-j

J		m, n	
input	evaluated	input	evaluated
-1	1/2	1	1/2
		-1	-1/2
-2	3/2	2	3/2
		1	1/2
		-1	-1/2
		-2	3/2
-3	5/2	3	5/2
		2	3/2
		1	1/2
		-1	-1/2
		-2	-3/2
		3	-5/2
-J	J - 1/2	J	J - 1/2
		⋮	⋮
		-j	-J + 1/2

Table 19-4 The correspondence between the integer J given as an argument *versus* the actual J value used in the function dJ.

Return Value:

A double precision number containing the specified reduced Wigner rotation matrix element, or a complex matrix containing the entire reduced Wigner rotation matrix.

Example(s):

```
#include <gamma.h>
main()
  {
  double r;                       // define a double precision number.
  r = dJ(2,-2,0,90.0);            // set r to d(2)2,-2(PI/2).
  r = dJ(-1,1,-1,90.0);           // re-set r to d(1/2)1/2,-1/2(PI/2).
  }
```

Mathematical Basis:

The reduced Wigner rotation matrix elements, $d^{(J)}_{m,n}(\beta)$ are given by[1]

$$d^{(J)}_{m,n}(\beta) = \sum_{t\ \text{integer}} K(m,n,t)[\sin(\beta/2)]^S[\cos(\beta/2)]^C \tag{8-8}$$

where the constant $K(m,n,t)$ is

$$K(m,n,t) = \frac{[(J+m)!(J-m)!(J+n)!(J-n)!]^{1/2}}{(J+m-t)!(J-n-t)!(t+n-m)!}\ , \tag{8-9}$$

and the coefficients S and C are

$$S(m,n,t) = 2t+n-m \quad \text{and} \quad C(m,n,t) = 2J+m-n-2t \tag{8-10}$$

respectively. The summation over t in (8-8) increments from zero over all non-negative factorials in the denominator. This equation gives for example the following table of elements for J = 3/2.

Reduced Rank 3/2 Wigner Rotation Matrix Elements

$$d^{(3/2)}_{m,n}(\beta)$$

m \ n	3/2	1/2	-1/2	-3/2
3/2	$\cos^3\left(\frac{\beta}{2}\right)$	$-\sqrt{3}\cos^2\left(\frac{\beta}{2}\right)\sin\left(\frac{\beta}{2}\right)$	$\sqrt{3}\cos\left(\frac{\beta}{2}\right)\sin^2\left(\frac{\beta}{2}\right)$	$-\sin^3\left(\frac{\beta}{2}\right)$
1/2	$\sqrt{3}\cos^2\left(\frac{\beta}{2}\right)\sin\left(\frac{\beta}{2}\right)$	$\cos\left(\frac{\beta}{2}\right)\left[3\cos^2\left(\frac{\beta}{2}\right)-2\right]$	$\sin\left(\frac{\beta}{2}\right)\left[3\sin^2\left(\frac{\beta}{2}\right)-2\right]$	$\sqrt{3}\cos\left(\frac{\beta}{2}\right)\sin^2\left(\frac{\beta}{2}\right)$
-1/2	$\sqrt{3}\cos\left(\frac{\beta}{2}\right)\sin^2\left(\frac{\beta}{2}\right)$	$-\sin\left(\frac{\beta}{2}\right)\left[3\sin^2\left(\frac{\beta}{2}\right)-2\right]$	$\cos\left(\frac{\beta}{2}\right)\left[3\cos^2\left(\frac{\beta}{2}\right)-2\right]$	$-\sqrt{3}\cos^2\left(\frac{\beta}{2}\right)\sin\left(\frac{\beta}{2}\right)$
-3/2	$\sin^3\left(\frac{\beta}{2}\right)$	$\sqrt{3}\cos\left(\frac{\beta}{2}\right)\sin^2\left(\frac{\beta}{2}\right)$	$\sqrt{3}\cos^2\left(\frac{\beta}{2}\right)\sin\left(\frac{\beta}{2}\right)$	$\cos^3\left(\frac{\beta}{2}\right)$

Table 19-5 The reduced Wigner rotation matrix elements of rank 3/2. The angle β is the standard

1. See Brink and Satchler, page 22, prior to equation (2.18).

Euler angle

The full Wigner rotation matrix elements are related to these *via*[1]

$$\boldsymbol{D}^{(J)}_{m,\,n}(\alpha,\,\beta,\,\gamma) \;=\; e^{-i\alpha m} d^{(J)}_{m,\,n}(\beta) e^{-i\gamma n} \;. \tag{8-11}$$

Other useful relationships concerning these elements are[2]

$$d^{(J)}_{m,\,n}(\beta) \;=\; (-1)^{m-n} d^{(J)}_{n,\,m}(\beta) \;=\; (-1)^{m-n} d^{(J)}_{-n,\,-m}(\beta) \quad . \tag{8-12}$$

This equation simplifies the previous given table for $d^{(3/2)}(\beta)$ down to a table of 6 unique elements.

Reduced Rank 3/2 Wigner Rotation Matrix Elements

$$d^{(3/2)}_{m,\,n}(\beta)$$

m \ n	**3/2**	**1/2**	**-1/2**	**-3/2**
3/2	$\cos^3\left(\frac{\beta}{2}\right)$	$-\sqrt{3}\cos^2\left(\frac{\beta}{2}\right)\sin\left(\frac{\beta}{2}\right)$	$\sqrt{3}\cos\left(\frac{\beta}{2}\right)\sin^2\left(\frac{\beta}{2}\right)$	$-\sin^3\left(\frac{\beta}{2}\right)$
1/2	$-d^{(3/2)}_{3/2,\,1/2}(\beta)$	$\cos\left(\frac{\beta}{2}\right)\left[3\cos^2\left(\frac{\beta}{2}\right)-2\right]$	$\sin\left(\frac{\beta}{2}\right)\left[3\sin^2\left(\frac{\beta}{2}\right)-2\right]$	$d^{(3/2)}_{3/2,\,-1/2}(\beta)$
-1/2	$d^{(3/2)}_{3/2,\,-1/2}(\beta)$	$-d^{(3/2)}_{1/2,\,-1/2}(\beta)$	$d^{(3/2)}_{1/2,\,1/2}(\beta)$	$d^{(3/2)}_{3/2,\,1/2}(\beta)$
-3/2	$d^{(3/2)}_{3/2,\,-3/2}(\beta)$	$d^{(3/2)}_{3/2,\,-1/2}(\beta)$	$-d^{(3/2)}_{3/2,\,1/2}(\beta)$	$d^{(3/2)}_{3/2,\,3/2}(\beta)$

Table 19-6 The reduced Wigner rotation matrix elements of rank 3/2 explicitly displaying the symmetry between the elements. There are 6 unique elements. The angle β is the standard Euler angle.

Examples of the $d^{(3/2)}(\beta)$ matrix are the following.

$$d^{(3/2)}\left(\frac{\pi}{2}\right) = \begin{bmatrix} 0 & 0 & 0 & -1 \\ 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \qquad d^{(3/2)}(\pi) = \frac{1}{2\sqrt{2}} \begin{bmatrix} 1 & -\sqrt{3} & -\sqrt{3} & -1 \\ \sqrt{3} & -1 & -1 & -\sqrt{3} \\ -\sqrt{3} & 1 & -1 & -\sqrt{3} \\ 1 & -\sqrt{3} & \sqrt{3} & 1 \end{bmatrix}$$

1. Brink and Satchler, page 22, equation (2.17).
2. Brink and Satchler, Appendix V, under Symmetry.

See Also:

8.5.6 DJ

Usage:

```
#include <spatial.h>
complex DJ(int J, int m, int n, double alpha, double beta, double gamma);
matrix DJ(int J, double alpha, double beta, double gamma);
```

Description:

The function DJ returns values for the rank J Wigner rotation matrix.

1. DJ(int J, int m, int n, double alpha, double beta, double gamma) - Returns the m, n element of the rank J Wigner rotation matrix for the Euler angles alpha, beta, and gamma.

$$\boldsymbol{D}^{(J)}_{m,n}(\alpha, \beta, \gamma) .$$

2. DJ(int J, double alpha, double beta, double gamma) - Returns the rank J Wigner rotation matrix for the Euler angles alpha, beta, and gamma.

$$\boldsymbol{D}^{(J)}(\alpha, \beta, \gamma)$$

The Euler angles are input in degrees. A negative J value is taken to imply a 1/2 integral unit of angular momentum.

Return Value:

A complex number containing the specified the Wigner rotation matrix element, or a complex matrix containing the entire Wigner rotation matrix.

Example(s):

```
#include <gamma.h>
main()
  {
  complex z;                          // define a general complex number.
  matrix D;                           // define a general matrix.
  z = DJ(2, 2, -1,90.0, 0.0, 90.0);   // set z to D2,2,-1(PI/2, 0, PI/2).
  D = DJ(1, 40.0, 180.0, 90.0);       // set matrix D to D1(40, PI, PI/2).
  }
```

Mathematical Basis:

The Wigner rotation matrix supplied by this function has it's elements computed from the following equation[1].

$$\boldsymbol{D}^{(J)}_{m,n}(\alpha, \beta, \gamma) = e^{-i\alpha m} d^{(J)}_{m,n}(\beta) e^{-i\gamma n} \tag{8-13}$$

In turn, the reduced Wigner rotation matrix elements, $d^{(J)}_{m,n}(\beta)$ are given by[2]

$$d^{(J)}_{m,n}(\beta) = \sum_{t} K(m, n, t)[\sin(\beta/2)]^{S}[\cos(\beta/2)]^{C} \quad , \tag{8-14}$$

where the constant $K(m, n, t)$ is

$$K(m, n, t) = \frac{[(J+m)!(J-m)!(J+n)!(J-n)!]^{1/2}}{(J+m-t)!(J-n-t)!(t+n-m)!} \tag{8-15}$$

and the coefficients S and C are

$$S(m, n, t) = 2t + n - m \quad \text{and} \quad C(m, n, t) = 2J + m - n - 2t \tag{8-16}$$

respectively. The summation over t in (8-8) increments from zero until over all non-negative factorials in the numerator. Also of some interest are the following relationships.

$$[\boldsymbol{D}^{(J)}_{m,n}(\alpha, \beta, \gamma)]^{*} = (-1)^{m-n} \boldsymbol{D}^{(J)}_{-m,-n}(\alpha, \beta, \gamma) = \boldsymbol{D}^{(J)}_{m,n}(-\alpha, -\beta, -\gamma) \tag{8-17}$$

See Also:

1. See Brink and Satchler, page 22, equation (2.17).
2. See Brink and Satchler, page 22, prior to equation (2.18).

8.6 Description

8.6.1 Introduction

The spatial functions library supplies normalized spherical harmonics, specified mathematically as

GAMMA Coordinate System

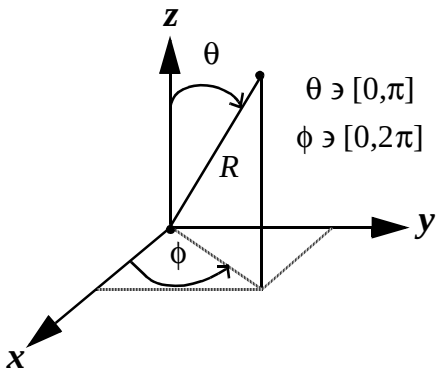


Figure 8-C GAMMA defined right handed Cartesian axes are shown as well as the spherical angles and radius.

8.6.2 Wigner J=1/2 Rotations

Reduced Rank 1/2 Wigner Rotation Matrix Elements

$d^{(1/2)}_{m,n}(\beta)$	$n = 1/2$	$m = -1/2$
$m = 1/2$	$\cos(\beta/2)$	$-\sin(\beta/2)$
$m = -1/2$	$\sin(\beta/2)$	$\cos(\beta/2)$

Table 19-7 : The reduced Wigner rotation matrix elements of rank 1/2.

The full Wigner rotation matrix elements are related to these *via*

$$\boldsymbol{D}^{(1/2)}_{m,n}(\alpha,\beta,\gamma) = e^{-i\alpha m} d^{(1/2)}_{m,n}(\beta) e^{-i\gamma n} \quad . \tag{8-18}$$

Other useful relationships concerning these elements are

$$d^{(1/2)}_{m,n}(\beta) = (-1)^{m-n} d^{(1/2)}_{n,m}(\beta) = (-1)^{m-n} d^{(1/2)}_{-n,-m}(\beta) \quad . \tag{8-19}$$

Examples of the $d^{(1/2)}_{m,n}(\beta)$ matrix are the following.

$$d^{(1/2)}_{m,n}\left(\frac{\pi}{3}\right) = \begin{bmatrix} 0.866 & -0.5 \\ 0.5 & 0.866 \end{bmatrix} \qquad d^{(1/2)}_{m,n}\left(\frac{\pi}{2}\right) = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 & -1 \\ 1 & 1 \end{bmatrix} \qquad d^{(1/2)}_{m,n}(\pi) = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}$$

Note that the above arrays correspond to $\boldsymbol{R}^{(1/2)}_{y}(\beta)$ and have the basis function ordering of that presented in the previous table.

See Also: d0, d1, d2, dJ

8.6.3 Wigner J=1 Rotations

The reduced (rank 1) Wigner rotation matrix elements supplied by this function are given in the following figure[1].

Reduced Rank 1 Wigner Rotation Matrix Elements

$d^{(1)}_{m,n}(\beta)$	$n = 1$	$n = 0$	$n = -1$
$m = 1$	$\cos^2(\beta/2)$	$-\sqrt{1/2}\sin\beta$	$\sin^2(\beta/2)$
$m = 0$	$\sqrt{1/2}\sin\beta$	$\cos\beta$	$-\sqrt{1/2}\sin\beta$
$m = -1$	$\sin^2(\beta/2)$	$\sqrt{1/2}\sin\beta$	$\cos^2(\beta/2)$

Figure 8-D The reduced Wigner rotation matrix elements of rank 1.

The relationship between these and the full Wigner rotation matrix elements as well as the symmetry expressions are given in the next expressions.

$$\begin{aligned} \boldsymbol{D}^{(1)}_{m,n}(\alpha,\beta,\gamma) &= e^{-i\alpha m} d^{(1)}_{m,n}(\beta) e^{-i\gamma n} \\ d^{(1)}_{m,n}(\beta) &= (-1)^{m-n} d^{(1)}_{n,m}(\beta) = (-1)^{m-n} d^{(1)}_{-n,-m}(\beta) \end{aligned} \tag{8-20}$$

From symmetry one may surmise that there are actually only four unique reduced rank 1 elements[2].

Reduced Rank 1 Unique Elements

$d^{(1)}_{m,n}(\beta)$	$n = 1$	$n = 0$	$n = -1$
$m = 1$	$\cos^2(\beta/2)$	$-\sqrt{1/2}\sin\beta$	$\sin^2(\beta/2)$
$m = 0$	$-d^{(1)}_{1,0}(\beta)$	$\cos\beta$	$d^{(1)}_{1,0}(\beta)$
$m = -1$	$d^{(1)}_{1,-1}(\beta)$	$-d^{(1)}_{1,0}(\beta)$	$d^{(1)}_{1,1}(\beta)$

Figure 8-E The reduced Wigner rotation matrix elements of rank 1 explicitly displaying the symmetry between the elements.

Examples of the $d^{(1)}_{m,n}(\beta)$ matrix are the following.

1. See Brink and Satchler, page 24, TABLE 1.
2. See Brink and Satchler, page 24, TABLE 1.

$$d^{(1)}_{m,n}\left(\frac{\pi}{3}\right) = \frac{1}{4}\begin{bmatrix} 3 & -2.565 & 1 \\ 2.565 & 2 & -2.565 \\ 1 & 2.565 & 3 \end{bmatrix} \quad d^{(1)}_{m,n}\left(\frac{\pi}{2}\right) = \frac{1}{2}\begin{bmatrix} 1 & -\sqrt{2} & 1 \\ \sqrt{2} & 0 & -\sqrt{2} \\ 1 & \sqrt{2} & 1 \end{bmatrix} \quad d^{(1)}_{m,n}(\pi) = \begin{bmatrix} 0 & 0 & 1 \\ 0 & -1 & 0 \\ 1 & 0 & 0 \end{bmatrix}$$

See Also:

8.6.4 Wigner J=1 Rotations

Reduced Rank 2 Wigner Rotation Matrix Elements

$$d^{(2)}_{m,n}(\beta)$$

m \ n	2	1	0	-1	-2
2	$\cos^4(\beta/2)$	$-\frac{1}{2}(1+\cos\beta)\sin\beta$	$\sqrt{3/8}\sin^2\beta$	$\frac{1}{2}(\cos\beta-1)\sin\beta$	$\sin^4(\beta/2)$
1	$\frac{1}{2}\sin\beta(\cos\beta+1)$	$\cos^2\beta-\frac{1}{2}(1-\cos\beta)$	$-\sqrt{3/2}\sin\beta\cos\beta$	$\frac{1}{2}(1+\cos\beta)-\cos^2\beta$	$\frac{1}{2}\sin\beta(\cos\beta-1)$
0	$\sqrt{3/8}\sin^2\beta$	$\sqrt{3/8}\sin(2\beta)$	$\frac{1}{2}(3\cos^2\beta-1)$	$-\sqrt{3/8}\sin(2\beta)$	$\sqrt{3/8}\sin^2\beta$
-1	$-\frac{1}{2}(\cos\beta-1)\sin\beta$	$\frac{1}{2}(1+\cos\beta)-\cos^2\beta$	$\sqrt{3/2}\sin\beta\cos\beta$	$\cos^2\beta-\frac{1}{2}(1-\cos\beta)$	$-\frac{1}{2}(1+\cos\beta)\sin\beta$
-2	$\sin^4(\beta/2)$	$-\frac{1}{2}(\cos\beta-1)\sin\beta$	$\sqrt{3/8}\sin^2\beta$	$\frac{1}{2}(1+\cos\beta)\sin\beta$	$\cos^4(\beta/2)$

Table 19-8 The reduced Wigner rotation matrix elements of rank 2. The angle β is the standard Euler angle

The full Wigner rotation matrix elements are related to these *via*

$$\boldsymbol{D}^{(2)}_{m,n}(\alpha,\beta,\gamma) = e^{-i\alpha m}d^{(2)}_{m,n}(\beta)e^{-i\gamma n} \quad . \tag{8-21}$$

Other useful relationships concerning these elements are

$$d^{(2)}_{m,n}(\beta) = (-1)^{m-n}d^{(2)}_{n,m}(\beta) = (-1)^{m-n}d^{(2)}_{-n,-m}(\beta) \quad . \tag{8-22}$$

This equation simplifies the above table of 25 elements down to a table containing only 9 unique components.

Reduced Rank 2 Wigner Rotation Matrix Elements

$$d^{(2)}_{m,n}(\beta)$$

m \ n	**2**	**1**	**0**	**-1**	**-2**
2	$\cos^4(\beta/2)$	$-\frac{1}{2}(1+\cos\beta)\sin\beta$	$\sqrt{3/8}\sin^2\beta$	$\frac{1}{2}(\cos\beta-1)\sin\beta$	$\sin^4(\beta/2)$
1	$-d^{(2)}_{2,1}(\beta)$	$\left(\cos\beta-\frac{1}{2}\right)(\cos\beta+1)$	$-\sqrt{3/2}\sin\beta\cos\beta$	$\left(\cos\beta+\frac{1}{2}\right)(1-\cos\beta)$	$d^{(2)}_{2,-1}(\beta)$
0	$d^{(2)}_{2,0}(\beta)$	$-d^{(2)}_{1,0}(\beta)$	$\frac{1}{2}(3\cos^2\beta-1)$	$d^{(2)}_{1,0}(\beta)$	$d^{(2)}_{2,0}(\beta)$
-1	$-d^{(2)}_{2,-1}(\beta)$	$d^{(2)}_{1,-1}(\beta)$	$-d^{(2)}_{1,0}(\beta)$	$d^{(2)}_{1,1}(\beta)$	$d^{(2)}_{2,1}(\beta)$
-2	$d^{(2)}_{2,-2}(\beta)$	$-d^{(2)}_{2,-1}(\beta)$	$d^{(2)}_{2,0}(\beta)$	$-d^{(2)}_{2,1}(\beta)$	$d^{(2)}_{2,2}(\beta)$

Table 19-9 The reduced Wigner rotation matrix elements of rank 2 explicitly displaying the symmetry between the elements. There are 9 unique elements. The angle β is the standard Euler angle.

Examples of the $d^{(2)}(\beta)$ matrix are the following.

$$d^{(2)}\left(\frac{\pi}{2}\right) = \frac{1}{2}\begin{bmatrix} \frac{1}{2} & -1 & \sqrt{3/2} & -1 & \frac{1}{2} \\ 1 & -1 & 0 & 1 & -1 \\ \sqrt{3/2} & 0 & -1 & 0 & \sqrt{3/2} \\ 1 & 1 & 0 & -1 & -1 \\ \frac{1}{2} & 1 & \sqrt{3/2} & 1 & \frac{1}{2} \end{bmatrix} \qquad d^{(2)}(\pi) = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{bmatrix}$$

See Also:

0.0.4 General Rotations

There are two viewpoints one may take in describing a general rotation, *active* and *passive*. An active rotation rotates the system whereas the passive rotation leaves the system stationary but rotates the coordinate system (axes). To demonstrate this pictorially, consider a rotation of the point P by an angle β in the xy-plane. In the active rotation view, the axes remain stationary while the coordinates of the point move relative to the original axes to produce a new point P´. Any function having a value at point P, f(P) prior to the rotation will have that value a point P´ after the rotation.

Active Rotations

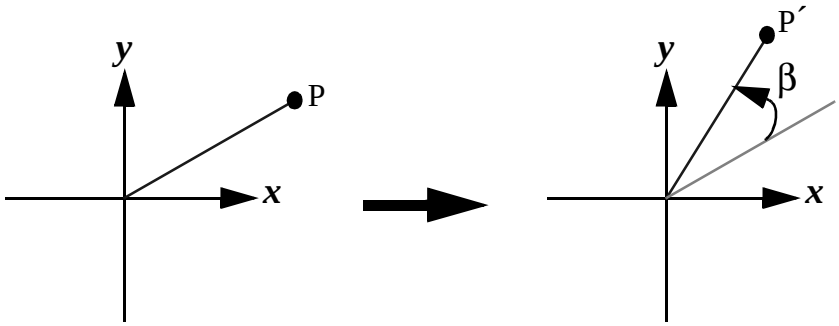


Figure 8-F A few of the many equations useful in determination of specific Wigner 3j coefficients.

A positive rotation is a positive rotation of the system, i.e. a counter-clockwise rotation of the system as shown in the above figure of the active rotation. Equal to this is passive view which is the negative (clockwise) rotation of the axes as shown in the figure below. Here the point P remains stationary while the coordinate system rotates by the same angle but in the opposite direction.

Passive Rotations

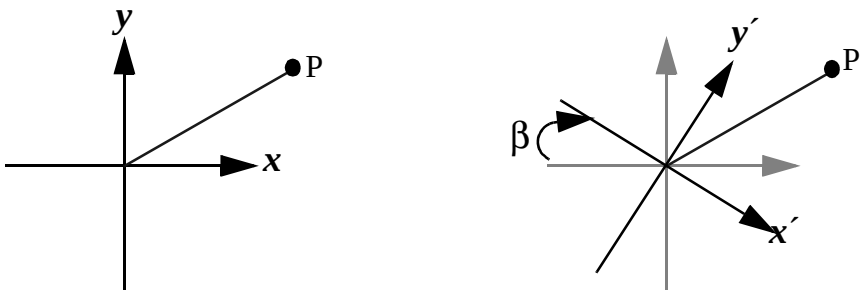


Figure 8-G A few of the many equations useful in determination of specific Wigner 3j coefficients.

We define a general operator to accomplish this rotation, **D**, where for a rotation about an arbitrary axis u we have

$$\boldsymbol{D}_u(\beta) \;=\; e^{-i\beta \boldsymbol{J}_u}\,. \tag{8-23}$$

0.0.5 Euler Angles

To produce an arbitrary rotation, that is a rotation about any arbitrary axis in coordinate space, it is convenient to specify three angles, the Euler angles $\{\alpha, \beta, \gamma\}$. Each rotation may be thought of as three secessive rotations about these angles as defined by the following.

$$\boldsymbol{D}_u(\alpha, \beta, \gamma) = e^{-i\gamma \boldsymbol{J}_z} e^{-i\beta \boldsymbol{J}_{y1}} e^{-i\alpha \boldsymbol{J}_z} . \tag{8-24}$$

$$\boldsymbol{D}_u(\alpha, \beta, \gamma) = e^{-i\alpha \boldsymbol{J}_z} e^{-i\beta \boldsymbol{J}_y} e^{-i\gamma \boldsymbol{J}_z} . \tag{8-25}$$

$$\langle Jm|\boldsymbol{D}(\alpha, \beta, \gamma)|Jn\rangle = \boldsymbol{D}^{(J)}_{m,n}(\alpha, \beta, \gamma) \quad . \tag{8-26}$$

This still needs some work *** ??? SOSI

9 Spatial Tensors

9.1 Overview

Class SPACE TENSOR (*space_T*) contains routines to handle general spatial tensors. Functions are provided herein to produce rank 1 and rank 2 spatial tensors from their respective components. In turn, these can be combined to form spatial tensors of higher rank. All spatial tensors can be rotated, multiplied with other spatial tensors, and combined with spin tensors to form Hamiltonians (general operators). Access to individual tensor elements is allowed and either the entire spin tensor or any element can be sent to standard output.

This class is analogous to *Class SPIN TENSOR* (*spin_T*) which treats tensors associated with spin angular momentum.

9.2 Available Functions

Basic Functions

space_T	- Spatial tensor constructor	page 145
=	- Spatial tensor assignment	page 145
+	- Spatial tensor addition	page 146
+=	- Spatial tensor unary addition	page 146
-	- Spatial tensor subtraction, negation	page 145
+=	- Spatial tensor unary subtraction	page 145

Spatial Tensor Functions with Scalars

*	- Scalar multiplication	

Spatial Tensor Principal Axis System Functions

alpha	- Spatial tensor alpha Euler angle value	
beta	- Spatial tensor beta Euler angle value	
gamma	- Spatial tensor gamma Euler angle value	

RANK 1 Spatial Tensor Functions

A1	- Rank 1 (spatial tensor, spherical components) from Cartesian components	
SphA1	- Rank 1 spatial tensor from spherical components	

RANK 2 Spatial Tensor Functions

A2	- Rank 2 (spatial tensor, spherical components) from Cartesian components	
PASys	- Principal axis system values: isotropic, delz, and eta	page 155
PASys_EA	- Principal axes Euler angles	page 155
iso	- Spatial tensor isotropic component (rank 2 tensors)	page 156
delz	- Spatial tensor delta z component (rank 2 tensors)	page 156
eta	- Spatial tensor eta value (rank 2 tensors)	page 157

Spatial Tensor Auxiliary Functions

exists	- Spatial tensor existence test	
rank	- Spatial tensor rank	
component	- Spatial tensor component access	page 158
T_mult	- Spatial tensor multiplication	page 159
rotate	- Spatial tensor rotations	page 159

Spatial Tensor Parameter Set and I/O Functions

=	- Spatial tensor parameter set assignment
+=	- Spatial tensor parameter set unary addition
write	- Write spatial tensor to disk file (as a parameter set).
read	- Read spatial tensor from disk file (from a parameter set).
<<	- Send spatial tensor to an output stream
Cartesian	- Send spatial tensor in Cartesian format to an output stream

9.3 Basic Routines

9.3.1 space_T

Usage:

```
#include <space_T.h>
space_T ()
space_T (space_T &SphT)
space_T (space_T &SphT, int l)
```

Description:

The function *space_T* is used to create a spatial tensor quantity. There are currently three methods for creating a new spatial tensor.

1. space_T () - When *space_T* is invoked without arguments it creates an empty spatial tensor.
2. space_T(space_T &SphT) - When *space_T* is used with another spatial tensor as its argument a new spatial tensor is produced that is identical to the spatial tensor given.
3. space_T(space_T &SphT, int l) - When *space_T* is used with another spatial tensor as well as an integer in the argument list, a new irreducible spatial tensor is produced that is identical to the irreducible spatial tensor of rank l included in the input spatial tensor.

The resultant spatial tensor can be used with all spatial tensor functions.

Return Value:

The function (constructor) *space_T* returns no values. It is used strictly to create a new spatial tensor.

Example(s):

```
#include <gamma.h>
main()
  {
  space_T SphT;                    // create an empty spatial tensor called SphT.
  SphT = A2(10.0,5.0,0.0);         // set SphT to a rank 2 symmetric tensor with eta=0.
  space_T SphT1(SphT);             // create a new spatial tensor SphT1 from SphT.
  space_T SphT2(SphT, 2);          // create irreducible tensor SphT2 from SphT rank 2.
  }
```

See Also: None

9.3.2 =

Usage:

```
#include <space_T.h>
space_T operator = (space_T &SphT);
```

Description:

The function (unary operator) = provides the use of equality in the algebraic manipulations of spatial tensors. The user may set one spatial tensor equal to another spatial tensor.

Return Value:

A spatial tensor is returned.

Example(s):

```
#include <gamma.h>
main()
  {
  space_T SphT;                    // create a new spatial tensor.
  SphT = A1(0.0,1.0.,0.0);         // set SphT to a rank 1 tensor, unit vector along y axis.
  space_T SphT1;                   // create another spatial tensor.
  SphT1 = SphT;                    // set SphT1 equal to SphT.
  }
```

See Also: None

9.3.3 +

Usage:

```
#include <space_T.h>
space_T operator + (space_T &SphT1, space_T &SphT2);
```

Description:

The function (unary operator) + provides the use of addition in the algebraic manipulations of spatial tensors. The user may set add any two spatial tensors. The resulting tensor will have a rank that is equal to that of the in input tensor with the highest rank.

Return Value:

A spatial tensor is returned.

Example(s):

```
#include <gamma.h>
main()
  {
  space_T SphT1, SphT2, SphT3;     // create three spatial tensors.
  SphT1 = A1(0.0,1.0.,0.0);        // SphT1 to a rank 1 tensor, unit vector along y axis.
  SphT2 = A2(10., 5., 7.2, 0., 0., 0.);  // set SphT1 to rank 2 symmetric tensor.
  SphT3 = SphT1 + SphT2;           // set SphT3 to sum of SphT2 and SphT1.
  }
```

See Also: +=

9.3.4 +=

Usage:

```
#include <space_T.h>
void operator += (space_T &SphT1, space_T &SphT2);
```

Description:

The function (unary operator) += provides the use of addition in the algebraic manipulations of spatial tensors. The user may set add any two spatial tensors. The resulting tensor will have a rank that is equal to that of the in input tensor with the highest rank.

Return Value:

None, the function is void.

Example(s):

```
#include <gamma.h>
main()
  {
  space_T SphT1, SphT2;                 // create two spatial tensors.
  SphT1 = A1(0.0,1.0.,0.0);             // set SphT1 as rank 1 tensor, unit vector along y axis.
  SphT2 = A2(10., 5., 7.2, 0., 0., 0.); // set SphT1 to rank 2 symmetric tensor.
  SphT1 += SphT2;                       // add SphT2 to SphT1. SphT1 now rank 2 also.
  }
```

See Also: +=

9.3.5 *

Usage:

```
#include <space_T.h>
space_T operator * (space_T &SphT, complex  &z);
space_T operator * (complex &z, space_T &SphT);
```

Description:

The function (unary operator) * provides the use scalar multiplication in the algebraic manipulations of spatial tensors. The user may multiply any spatial tensors by a complex number. The resulting tensor will have all components multiplied by the scalar.

Return Value:

A spatial tensor is returned.

Example(s):

```
#include <gamma.h>
main()
  {
  space_T SphT1, SphT2;                 // create two spatial tensors.
  complex z;                            // create a complex number.
  z = complex(0., 5.);                  // set z to 0+5i.
  SphT1 = A1(0.0,1.0.,0.0);             // SphT1 to a rank 1 tensor, unit vector along y axis.
  SphT2 = z*SphT1;                      // set SphT2 to SphT1 times 5i.
  }
```

See Also: *=, T_mult, T_prod

9.3.6 *=

Usage:

```
#include <space_T.h>
void operator *= (space_T &SphT, complex  &z);
```

Description:

The function (unary operator) *= provides the use scalar multiplication in the algebraic manipulations of spatial

tensors. The user may multiply any spatial tensors by a complex number. The tensor will have all components multiplied by the scalar.

Return Value:

None, the function is void. The assigned spatial tensor is modified.

Example(s):

```
#include <gamma.h>
main()
  {
  space_T SphT1;                    // create a spatial tensor.
  complex z;                        // create a complex number.
  z = complex(0., 5.);              // set z to 0+5i.
  SphT1 = A1(0.0,1.0.,0.0);         // SphT1 to a rank 1 tensor, unit vector along y axis.
  SphT1 *= z;                       // set SphT1 to SphT1 times 5i.
  }
```

See Also: ***, T_mult, T_prod

9.4 Rank 1 Tensors

9.4.1 A1

Usage:

```
#include <space_T.h>
space_T A1(coord& pt);
space_T A1(double x, double y, double z);
complex A1(coord& pt, int m);
complex A1(double x, double y, double z, int m);
```

Description:

Given the **Cartesian components,** the function *A1* returns a rank 1 spatial tensor or a specified irreducible spherical component of that tensor.

1. A1 (coord& pt) - When *A1* is invoked with a coordinate in the argument it creates the standard irreducible spherical rank 1 spatial tensor[1] assuming the coordinate is Cartesian.
2. A1 (double x, double y, double z) - When *A1* is invoked with x, y, and z coordinates in the argument it creates the standard irreducible spherical rank 1 spatial tensor assuming x, y, and z form a Cartesian coordinate.
3. A1(coord& pt, int spin, int m) - When *T1* is invoked with a coordinate and angular momentum components the function returns a complex number which is the l, m irreducible spherical component of the standard rank 1 spatial tensor.[2] It assumes the input Coordinate is Cartesian.
4. A1(double x, double y, double z, int spin, int m) - When *T1* is invoked with x, y, and z coordinates and angular momentum components the function returns a complex number which is the l, m irreducible spherical component of the standard rank 1 spatial tensor. It assumes x, y, and z form a Cartesian coordinate.

The rank 1 spatial tensor consists of three irreducible spherical components specified by $A_{l,m}$ where $m \in [-1, 1]$. All spatial tensor components are returned as complex numbers.

Return Value:

A rank 1 irreducible spherical spatial tensor.

Example(s):

```
#include <gamma.h>
main()
  {
  space_T SphT;                    // create a general spatial tensor called SphT.
  coord pt(0.0,1.0,0.0);           // create a coordinate point, unit y-vector.
  SphT = A1(0.0, 0.0, 1.0);        // SphT is rank 1 spatial tensor for the unit z-vector.
```

1. This function originally took a vector as well, the first three real values coinciding with x, y, and z, but this is no longer supported. It is marked for removal but may still work.
2. The above footnote applies here also. Originally, l=0 was an option but this is no longer supported. It is marked for removal but may work by adding 0 as a additional final argument in the function call.

```
SphT = A1(pt);                  // SphT is rank 1 spatial tensor for the unit y-vector.
cout << A1(pt,1);               // outputs the m=1 component of A1, here -i/(√2) .
cout << A1(0.0,0.0,1.0,1);      // outputs m=1 component of A1, in this case zero.
}
```

Mathematical Basis:

There are three irreducible spherical components of a rank 1 tensor. For a spatial rank 1 tensor these components are explicitly given by equation (9-5).

$$A_{1,0} = v_z \qquad A_{1,\pm 1} = \mp\frac{1}{\sqrt{2}}[v_x \pm i v_y] = \mp\frac{1}{\sqrt{2}} v_{\pm} \ ,$$

where ν is a vector with coordinates referenced to an arbitrary Cartesian axis system,

$$\boldsymbol{v} = v_x \boldsymbol{i} + v_y \boldsymbol{j} + v_z \boldsymbol{k} \, .$$

The function *A1* will produce a spatial tensor with these components. Note that the input coordinates (coordinate point or x, y, z) are assumed to be Cartesian and referenced to an arbitrary axis system, *i.e.* the user must externally keep track of which axis system the tensor is expressed in.

See Also: SphA1, A2

9.4.2 SphA1

Usage:

```
#include <space_T.h>
space_T sphA1(coord& pt);
space_T sphA1(complex v1, complex v0, complex vm1);
```

Description:

The function *sphA1* returns a spherical irreducible rank 1 spatial tensor with **components as given** in the function argument list.

1. sphA1 (coord& pt) - When *sphA1* is invoked with a coordinate in the argument it returns an irreducible spherical rank 1 spatial tensor[1] with components equivalent to the input coordinate components.
2. sphA1 (complex v1, complex v0, complex vm1) - When *sphA1* is invoked with v1, v0, and vm1 complex values in the argument it creates an irreducible spherical rank 1 spatial tensor with components equivalent to the input values.

The rank 1 spatial tensor consists of three irreducible spherical components specified by $\boldsymbol{A_{l,m}}$ where $\boldsymbol{m \in [-1, 1]}$.

Return Value:

A rank 1 irreducible spherical spatial tensor.

1. This function originally took a vector as well, the first three real values coinciding with plus, zero, and minus, but this is no longer supported. It is marked for removal but may still work.

Example(s):

```
#include <gamma.h>
main()
  {
  space_T SphT;                    // create a general spatial tensor called SphT.
  coord pt(0.0, 1.0, 0.0);         // create a pt, keep in mind this is used as spherical.
  SphT = SphA1(0.0, 1.0, 0.0);     // SphT is rank 1 spatial tensor for the unit z-vector.
  SphT = SphA1(pt);                // SphT is rank 1 spatial tensor for the unit z-vector.
  }
```

Mathematical Basis:

There are three irreducible spherical components of a rank 1 tensor. For a spatial rank 1 tensor these components are given by $A_{1,1}$, $A_{1,0}$, and $A_{1,-1}$, the three values taken from the function arguments of SphA1. Note that the input coordinates (coordinate point or plus, zero, minus) are assumed to be spherical and referenced to an arbitrary axis system, *i.e.* the user must externally keep track of which axes the tensor is expressed in. See equation (9-5) on page 167 for the relationship between the spherical components and the Cartesian components.

See Also: A1, A2

9.5 Rank 2 Tensors

9.5.1 A2

Usage:

```
#include <space_T.h>
space_T A2(matrix& mx, double alpha=0, double beta=0, double gamma=0);
space_T A2(matrix& mx, coord& EA);
space_T A2(double Aiso, double delzz, double eta=0,
                                    double alpha=0, double beta=0, double gamma=0);
space_T A2(coord& comps, double alpha=0, double beta=0, double gamma=0);
space_T A2(coord& comps, coord& angles);
```

Description:

The function *A2* returns either an entire rank 2 spatial tensor or a specific component of that tensor.

1. A2 (matrix& mx, double alpha=0, double beta=0, double gamma=0) - When *A2* is invoked with a matrix and three Euler angles (alpha, beta, and gamma) it creates the standard irreducible spherical rank 2 spatial tensor maintaining the angles as those which will rotate it to its reference axes. The matrix must be 3x3 and referenced to some Cartesian axes. (If the matrix is complex its real values will be utilized).
2. A2 (matrix& mx, coord EA) - When *A2* is invoked with three tensors values (Aiso, delzz, eta) and three Euler angles (alpha, beta, and gamma) it creates the standard irreducible spherical rank 2 spatial tensor.
3. A2 (double Aiso, double delzz, double eta=0, double alpha=0, double beta=0, double gamma=0) - When *A2* is invoked with three tensors values (Aiso, delzz, eta) and three Euler angles (alpha, beta, and gamma) it creates the standard irreducible spherical rank 2 spatial tensor.
4. A2(coord& comps, double alpha=0, double beta=0, double gamma=0) - When *A2* is invoked with a spin system, spin indices, and angular momentum components the function returns a spin operator which is the l, m irreducible spherical component of the standard rank 2 spatial tensor.

A general rank 2 tensor consists of nine irreducible spherical components. For the spatial rank 2 tensor these are specified as $A_{l,m}$ where $l \in [0, 2]$ and $m \in [-l, l]$ All tensor components are returned as complex numbers.

Return Value:

A rank 2 irreducible spherical spatial tensor.

Example(s):

```
#include <gamma.h>
main()
  {
  spin_system AMX(3);          // create a three spin system called AMX.
  spin_op SOp(AMX);            // create spin operator associated with AMX, SOp.
  space_T SphT(AMX);           // create spatial tensor associated with AMX, SphT.
  SphT = T1(AMX, 0);           // SphT is the rank 1 spatial tensor for spin 0.
  SOp = T1(AMX, 0, 1, 0);      // SOp is 1,0 rank 1 spatial tensor component spin 0.
```

```
cout << SOp;                    // prints SOp, 1,1 component of the spatial tensor.
}
```

Mathematical Basis:

In general, a rank 2 tensor (reducible) contains a possible nine components. These tensor components are internally maintained irreducible spherical format[1] which, for the spatial tensor, are specified as $\boldsymbol{A_{l,m}}$. This relationship between the general (reducible) rank 2 tensor $\boldsymbol{A_2}$, its associated irreducible tensors $\boldsymbol{A_l}$ and their components $\boldsymbol{A_{l,m}}$ is depicted in the following figure.

REDUCIBLE RANK 2 TENSOR	IRREDUCIBLE TENSORS	IRREDUCIBLE SPHERICAL COMPONENTS
	$\mathbf{A}_0$	$\mathbf{A}_0: A_{00}$
$\mathbf{A}_2$	$\mathbf{A}_1$	$\mathbf{A}_1: A_{11}\ A_{10}\ A_{1\text{-}1}$
	$\mathbf{A}_2$	$\mathbf{A}_2: A_{22}\ A_{21}\ A_{20}\ A_{2\text{-}1}\ A_{2\text{-}2}$

In contrast to this internal storage format, this function assumes that the tensor is being input in terms of its Cartesian components and will automatically convert the given values into the analogous irreducible spherical values. This is depicted in the next diagram. The input tensor is, in the most general case, a 3x3 matrix or in the case of a symmetric tensor defined by three values $\boldsymbol{A_{iso}}$, δ_{zz}, and η.

INPUT TENSOR STORED TENSOR

$$\boldsymbol{A}_{2Cart} = \begin{bmatrix} A_{xx} & A_{xy} & A_{xz} \\ A_{yx} & A_{yy} & A_{yz} \\ A_{zx} & A_{zy} & A_{zz} \end{bmatrix} \longrightarrow \begin{array}{l} \mathbf{A}_0: A_{00} \\ \mathbf{A}_1: A_{11}\ A_{10}\ A_{1\text{-}1} \\ \mathbf{A}_2: A_{22}\ A_{21}\ A_{20}\ A_{2\text{-}1}\ A_{2\text{-}2} \end{array}$$

$$\boldsymbol{A}_{2Cart,\,Sym} = A_{iso}\hat{I} + \delta_{zz}\begin{bmatrix} -\frac{1}{2}(1-\eta) & 0 & 0 \\ 0 & -\frac{1}{2}(1+\eta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \longrightarrow \begin{array}{l} \mathbf{A}_0: A_{00} \\ \mathbf{A}_1: 0 \\ \mathbf{A}_2: A_{22}\ A_{21}\ A_{20}\ A_{2\text{-}1}\ A_{2\text{-}2} \end{array}$$

For treatment of the general case, the Cartesian tensor is input simply as a (3x3) matrix and the function uses the nine formulas given in (9-6) to determine the irreducible spherical components from the Cartesian components[2].

1. See sosi on sosi as well its surrounding text for a description of the general rank 2 tensor relative to the irreducible spherical components.
2. These formulas are in agreement with Mehring, page 14, Table 2.2.

$$A_{0,0} = \frac{-1}{\sqrt{3}}[A_{xx} + A_{yy} + A_{zz}] = \frac{-1}{\sqrt{3}}Tr\{\boldsymbol{A}\}$$

$$A_{1,0} = \frac{-i}{\sqrt{2}}[A_{xy} - A_{yx}] \qquad A_{1,\pm1} = \frac{-1}{2}[(A_{zx} - A_{xz}) \pm i(A_{zy} - A_{yz})]$$

$$A_{2,0} = \frac{1}{\sqrt{6}}[-A_{xx} - A_{yy} + 2A_{zz}] = \frac{1}{\sqrt{6}}[3A_{zz} - Tr\{\boldsymbol{A}\}]$$

$$A_{2,\pm1} = \mp\frac{1}{2}[(A_{xz} + A_{zx}) \pm i(A_{yz} + A_{zy})] \qquad A_{2,\pm2} = \frac{1}{2}[(A_{xx} - A_{yy}) \pm i(A_{xy} + A_{yx})]$$

Typically, one knows only about the tensor components when $\boldsymbol{A_2}$ is expressed relative to its principal axis system (PAS), that is, with respect to a set of Cartesian axes where $\boldsymbol{A_2}$ appears diagonal[1]. When the tensor is symmetric, and this is usually the case, only three rather than nine components are needed to describe the tensor. Rather than use $\boldsymbol{A_{xx}}$, $\boldsymbol{A_{yy}}$, and $\boldsymbol{A_{zz}}$ it is more convenient to use the quantities $\boldsymbol{A_{iso}}$, $\delta_{\boldsymbol{zz}}$, and η. These three values can be input to formulate a rank two spatial tensor according to equation (9-4).

$$\left[A_{iso}\ \delta_{zz}\ \eta\right] \rightarrow A_{iso} + \delta_{zz}\begin{bmatrix} -\frac{1}{2}(1-\eta) & 0 & 0 \\ 0 & -\frac{1}{2}(1+\eta) & 0 \\ 0 & 0 & 1 \end{bmatrix} = \boldsymbol{A}_{2Cart,\,Sym}$$

The spherical irreducible components are then determined from the equations labeled (9-7).

$$A_{0,0}(PAS)_{Sym} = -\sqrt{3}A_{iso}$$

$$A_{1,0}(PAS)_{Sym} = A_{1,\pm1}(PAS)_{Sym} = 0$$

$$A_{2,0}(PAS)_{Sym} = \sqrt{3/2}\delta_{zz} \qquad A_{2,\pm1}(PAS)_{Sym} = 0$$

$$A_{2,\pm2}(PAS)_{Sym} = \frac{1}{2}\delta_{zz}\eta$$

When the tensor is not symmetric these three components {$\boldsymbol{A_{iso}}$, $\delta_{\boldsymbol{zz}}$, and η} and the three rank one components are necessary to completely specify the tensor. These six values can be input to formulate a rank two spatial tensor according to equations (9-2) and (9-3).

1. If not symmetric the PAS system are the axes in which the irreducible $\boldsymbol{A_2}$ is diagonal. See the Discussion section at the end of this chapter.

$$\left[\alpha_{xy}\ \alpha_{yz}\ \alpha_{zx}\ A_{iso}\ \delta_{zz}\ \eta\right] \rightarrow A_{iso}I + \begin{bmatrix} 0 & \alpha_{xy} & -\alpha_{zx} \\ -\alpha_{xy} & 0 & \alpha_{yz} \\ \alpha_{zx} & -\alpha_{yz} & 0 \end{bmatrix} + \delta_{zz}\begin{bmatrix} -\frac{1}{2}(1-\eta) & 0 & 0 \\ 0 & -\frac{1}{2}(1+\eta) & 0 \\ 0 & 0 & 1 \end{bmatrix} = \boldsymbol{A}_2(PAS)_{CART}$$

The spherical irreducible components in this situation are then determined from the equations labeled (9-7) on page 168.

$$A_{0,0}(PAS) = -\sqrt{3}A_{iso}$$

$$A_{1,0}(PAS) = -i\sqrt{2}\sigma_{xy} \qquad A_{1,\pm1}(PAS) = \sigma_{xz} \pm i\sigma_{yz}$$

$$A_{2,0}(PAS) = \sqrt{3/2}\delta_{zz} \qquad A_{2,\pm1}(PAS) = 0 \qquad A_{2,\pm2}(PAS) = \frac{1}{2}\delta_{zz}\eta$$

See Also: A1

9.5.2 PASys

Usage:

```
#include <space_T.h>
coord space_T::PASys ();
```

Description:

Defined only for a rank 2 tensor, this function returns the values used to define the symmetric tensor in the principal axis system, {$\boldsymbol{A_{iso}}$, δ_{zz}, and η}.

Return Value:

A coordinate is returned.

Example:

```
#include <gamma.h>
main()
  {
  }
```

See Also: PAS_EA, iso, delz, eta

9.5.3 PASys_EA

Usage:

```
#include <space_T.h>
coord space_T::PASys_EA ();
```

Description:

Defined only for a rank 2 tensor, this function returns the three angles used to define the principal axes relative to the current tensor axes, { α, β, γ }.

Return Value:

A coordinate is returned.

Example:

```
#include <gamma.h>
main()
  {
  }
```

See Also: PAS_EA, iso, delz, eta

9.5.4 iso

Usage:

```
#include <space_T.h>
double space_T::iso();
void space_T::iso(double Aiso);
```

Description:

Defined only for a rank 2 tensor, this function either returns the isotropic value of the spatial tensor as defined by equation (9-1)

$$A_{iso} = \frac{1}{3} Tr(\boldsymbol{A}_2) \ ,$$

or sets the isotropic value. Setting this value has no effect on other spherical tensor elements.

Return Value:

A double precision number is returned when requesting the value, the function is void when setting the value.

Examples:

```
#include <gamma.h>
main()
  {
  space_T SphT;                          // create two new empty spatial tensors.
  SphT = A2(7.2, 50.0);                  // SphT now rank 2, Aiso=7.2 PPM, delz=50.0.
  SphT.iso(2.3);                         // SphT now rank 2, Aiso=2.3 PPM, delz=50.0.
  cout << "\nAiso = " << SphT.iso();     // Output the isotropic value.
  }
```

See Also: delz, eta

9.5.5 delz

Usage:

```
#include <space_T.h>
double space_T::delz ();
```

Description:

Defined only for a rank 2 tensor, this function returns the delta z value of the spatial tensor as defined by equation (9-4)

$$\delta_{zz} = A_{zz} - \frac{1}{3}Tr(\boldsymbol{A}_2) = A_{zz} - A_{iso} .$$

Return Value:

A double precision number is returned.

Example:

```
#include <gamma.h>
main()
  {
  space_T SphT;                    // create two new empty spatial tensors.
  ***;                             // .
  cout << "\ndelz = " << SphT.delz(); // Output the delta z value.
  }
```

See Also: iso, eta

9.5.6 eta

Usage:

```
#include <space_T.h>
double space_T::eta();
```

Description:

Defined only for a rank 2 tensor, this function returns the eta value of the spatial tensor as defined by equation (9-4)

$$\eta = (\delta_{xx} - \delta_{yy})/\delta_{zz} = \frac{(A_{xx} - A_{iso}) - (A_{yy} - A_{iso})}{(A_{zz} - A_{iso})} = \frac{(A_{xx} - A_{yy})}{(A_{zz} - A_{iso})}.$$

Return Value:

A double precision number is returned.

Example:

```
#include <gamma.h>
main()
  {
  space_T SphT;                    // create two new empty spatial tensors.
  ***;                             // .
  cout << "\neta = " << SphT.eta(); // Output the eta value.
  }
```

See Also: iso, delz

9.6 Complex Routines

9.6.1 component

Usage:

```
#include <space_T.h>
space_T space_T::component (int l);
complex space_T::component (int l, int m);
```

Description:

The member function *component* provides direct access to the individual irreducible spherical components of the input spatial tensor.

1. component (int l) - When invoked with only a rank index as arguments the function *component* returns an irreducible spatial tensor equivalent to the irreducible rank l component of the input tensor.
2. component (int l, int m) - When invoked with a rank index and a component index as arguments the function *component* returns a complex number which is the irreducible spherical l, m component of the input tensor.

A general rank ***l*** tensor can be broken up into irreducible tensors of rank ***l***, ***l-1,...., 0***. Each of these tensors has components which are stored internally as irreducible spherical components,

$$A_{l,\,m}\,.$$

This returns either one of the irreducible tensors of specified rank which makes up the input tensor or one of the irreducible tensor components. The tensor indices should not lie outside of their intrinsic range as set by the tensor rank, namely $\boldsymbol{l} \in \boldsymbol{[0,\ k]}$ and $\boldsymbol{m} \in \boldsymbol{[-l,\ l]}$ where ***k*** is the rank of the tensor. If the tensor itself is irreducible only the components with ***l=k*** will exist and the function returns zero if asked for components with ***l<k***.

Return Value:

A complex number or an irreducible spatial tensor is returned.

Example(s):

```
#include <gamma.h>
main()
  {
  space_T SphT, sphT1;                    // create two new empty spatial tensors.
  complex z;                              // create a new complex number.
  SphT = A1(0.0, 0.0, -2);                // SphT to rank 1 tensor, length 2 vector along -z axis.
  SphT1=SphT.component(0.0, 0.0, -2); // SphT1 irreducible rank 1 tensor 0f SphT (equal).
  z = SphT.component(1,-1);               // set z to the 1,-1 component of the tensor SphT.
  //z = SphT.component(2,0);              // access rank 2 comp. of a rank 1 tensor! Fatal error.
  }
```

See Also: None

9.6.2 rotate

Usage:

#include <gamma.h>
space_T space_T::rotate = (double alpha, double beta, double gamma);
space_T space_T::rotate = (coord &EA);

Description:

The member function *rotate* rotates the spatial tensor coordinate system by Euler angles alpha, beta, and gamma to produce a new spatial tensor.

1. rotate(double alpha, double beta, double gamma) - When rotate is invoked with a these arguments it rotates the coordinate system to which the tensor is referenced to by the Euler angles alpha, beta, and gamma. The angles are input in degrees
2. rotate(coord &EA) - This use performs identically to 1. except that the three Euler angles alpha, beta, and gamma are contained in a single coordinate EA.

Return Value:

A new spin tensor is returned.

Example(s):

```
#include <gamma.h>
main()
  {
  spin_system AX(2);                  // create a two spin system called AX.
  space_T SphT(AX);                   // new spin tensor associated with spin system AX.
  SphT = A1(0.0, 0.0, -2);            // SphT to rank 1 tensor, length 2 vector along -z axis.
  SphT = SphT.rotate(0, 90.0, 0);     // rotate coordinate axes of SphT 90 degrees about y.
  coord EA(0, 90.0, 0);               // define a coordinate point with Euler angles.
  SphT = SphT.rotate(EA);             // another rotation of SphT 90 degrees about y.
  }
```

See Also: None

Mathematical Basis:

9.6.3 T_mult

Usage:

#include <space_T.h>
space_T T_mult (space_T &SphT1, space_T &SphT2);
space_T T_mult (space_T &SphT1, intl1, space_T &SphT2, int l2);
space_T T_mult (space_T &SphT1, intl1, space_T &SphT2, int l2, int L);
complex T_mult (space_T &SphT1, intl1, space_T &SphT2, int l2, int L, int M);

Description:

The function *T_mult* multiplies two tensors (cross product) together to produce a new tensor.

$$\boldsymbol{A = A1 \times A2}$$

The user can obtain the entire resulting tensor, the tensor resulting from the product with A1 and A2 irreducible,

an irreducible tensor component of such a product or a specific component of this latter product.

1. T_mult (space_T &SphT1, space_T &SphT2) - Returns a spatial tensor which is the product of the two input spatial tensors. The resulting tensor will have a rank which is the sum of the individual tensor ranks and will be stored in terms of its irreducible spherical components.
2. T_mult (space_T &SphT1, int l1, space_T &SphT2, int l2) - Returns a spatial tensor which is the product of the rank l1 irreducible component of T1 and the rank l2 irreducible component of T2.The resulting tensor will have a rank which is the sum of l1 and l2 and will be stored in terms of its irreducible spherical components.
3. T_mult (space_T &SphT1, int l1, space_T &SphT2, int l2, int L) - Returns an irreducible spatial tensor which is the rank L component arising from the tensor product of the rank l1 irreducible component of T1 and the rank l2 irreducible component of T2. The resulting tensor will have a rank L, be irreducible and have its components stored spherical.
4. T_mult (space_T &SphT1, int l1, space_T &SphT2, int l2, int L, int M) - Returns a complex number which is the L,M component of the spatial tensor which is the product of the rank l1 irreducible component of T1 and the rank l2 irreducible component of T2.

Return Value:

Either a spatial tensor or a complex number is returned.

Example(s):

```
#include <gamma.h>
main()
  {
  space_T SphT;                          // create a new empty spatial tensor.
  SphT = sphA1(Y11(90., 45.), Y10(90., 45,), Y1m1(90.,45.); // SphT rank 1 harmonics.
  space_T SphT1(SphT);                   // create a new spatial tensor identical to SphT.
  space_T SphT2;                         // create a new empty spatial tensor.
  SphT2 = T_prod(SphT, SphT1);           // set SphT2 equal to the product SphT x SphT1, Y2's
  SphT1 = T_prod(SphT, SphT2);           // set SphT1 equal to the product SphT x SphT2, Y3's
  }
```

See Also: None

Mathematical Basis:

The product of two irreducible spherical spatial tensors, ***A1*** and ***A2*** produces another spatial tensor, ***A***, according the equation ***A = A1 x A2***. The irreducible spherical components of ***A*** are given by equation (9-15)[1]

$$A_{L,M} = \sum_{m1}^{\pm l1} \sum_{m2}^{\pm l2} A1_{l1,m1} A2_{l2,m2} \langle l1 l2 m1 m2 | LM \rangle$$

where $L \in [l1+l2, |l1-l2|]$, $M = m1 + m2$ and $\langle l1 l2 m1 m2 | LM \rangle$ are Clebsch-Gordan coefficients. The entire irreducible rank ***L*** tensor is returned from usage 3. of the function and the ***L, M*** component from usage 4, $A_{L,M}$. When the product of two reducible tensors is per-

1. This formula is found in Brink and Satchler, page 52, equation (4.6).

formed, each of the input tensors must be expanded in terms of irreducible components.

9.7 Parameters and I/O

Usage:

```
#include <space_T.h>
void operator = (p_set& pset, const space_T& SphT);
```

Description:

The operator = provides users with the capability to store spatial tensors as a parameter set.

Return Value:

A parameter set containing (only) the information concerning the spatial tensor.

Example:

```
#include <gamma.h>
main()
  {
  }
```

See Also:

9.7.1 +=

Usage:

```
#include <space_T.h>
void operator += (p_set& pset, const space_T& SphT);
```

Description:

The operator += provides users with the capability to store spatial tensors with parameter sets. Unlike the assignment, this adds the tensor information to the existing values in the parameter set.

Return Value:

A parameter set additionally containing the information concerning the spatial tensor.

Example:

```
#include <gamma.h>
main()
  {
  }
```

See Also:

9.7.2 write

Usage:

```
#include <space_T.h>
void space_T::write (const String& filename);
```

Description:

The function write allows users to write a tensor directly to an output file in readable ASCII format (as a parameter).

Return Value:

None

Example:

```
#include <gamma.h>
main()
  {
  }
```

See Also: read

9.7.3 read

Usage:

```
#include <space_T.h>
void space_T::read (const String& filename);
```

Description:

The function read allows users to read a tensor directly to an input file in readable ASCII format (as a parameter).

Return Value:

None

Example:

```
#include <gamma.h>
main()
  {
  }
```

See Also: read

9.7.4 <<

Usage:

```
#include <space_T.h>
ostream& << (ostream& ostr, const space_T& SphT);
```

Description:

The operator << provides standard output abilities for spatial tensors.

Return Value:

An ostream containing all irreducible spherical components of the spatial tensor specified.

Example:

```
#include <gamma.h>
main()
  {
  space_T SphT;                    // create an empty spatial tensor called SphT.
  SphT = A2(10.0, 25.0, 50.0);     // set SphT to some rank two tensor.
  cout << SphT;                    // print SphT. Rank 1 won't print (0 symmetric A2).
  }
```

See Also: Cartesian

9.7.5 Cartesian

Usage:

```
#include <space_T.h>
void Cartesian (const space_T& SphT);
```

Description:

The function Cartesian attempts to output the Cartesian components of the input tensor. This currently works for only up to a rank 2 tensor, higher ranks are not supported.

Return Value:

None. Prints the Cartesian tensor components to standard output.

Example(s):

```
#include <gamma.h>
main()
  {
  space_T SphT;                    // create an empty spatial tensor called SphT.
  SphT = A2(10.0, 25.0, 50.0);     // set SphT to some symmetric rank two tensor.
  Cartesian(cout, SphT);           // prints the Cartesian components of the tensor.
  }
```

See Also: <<

9.8 Description

9.8.1 Introduction

Class SPATIAL TENSOR contains all the properties of tensors specifically associated with spatial variables. The class contains routines to handle general spatial tensors, each spatial tensor is maintained in its irreducible spherical form, each component being a complex number.

Coordinate Axes

Each spatial tensor has an internal reference to some arbitrary set of axes. These is constructed.tains all the properties of tensors specifically associated with spatial variables.

9.8.2 Cartesian Tensors

Cartesian tensors are those which are referenced to a Cartesian coordinate system, for example the ***x***, ***y*** and ***z*** axes. These are depicted in the next figure for ranks 0, 1, and 2 .

Cartesian Tensors

Rank 0 Rank 1 Rank 2

$$\boldsymbol{A}_0 = A \qquad \boldsymbol{A}_{1_{Cart}} = \begin{bmatrix} A_x \\ A_y \\ A_z \end{bmatrix} \qquad \boldsymbol{A}_{2_{Cart}} = \begin{bmatrix} A_{xx} & A_{xy} & A_{xz} \\ A_{yx} & A_{yy} & A_{yz} \\ A_{zx} & A_{zy} & A_{zz} \end{bmatrix}$$

Figure A Depiction of Cartesian tensors of ranks zero, one, and two. For a spatial tensor the tensor elements will be numbers.

A rank 0 tensor just a scalar, a simple number. There is only one component (which is actually independent of coordinate system). A rank 1 tensor is a vector. The three components can be described in typical vector nomenclature, referenced to some arbitrary set of Cartesian axes (Cart). A general rank 2 tensor contains nine components. This can be represented as a simple matrix (3x3) matrix, again with components referenced to some arbitrary set of Cartesian axes.

Unlike the rank 0 and rank 1 tensors, tensors of rank J (J>1) can be broken down into tensors with ranks spanning 0 to J. For example, $\boldsymbol{A_2}$ as depicted in the previous figure can be broken up into a sum of three tensors of ranks 0 through 2, a scalar, vector and a matrix respectively.

$$\boldsymbol{A}_{2_{Cart}} = \begin{bmatrix} A_{xx} & A_{xy} & A_{xz} \\ A_{yx} & A_{yy} & A_{yz} \\ A_{zx} & A_{zy} & A_{zz} \end{bmatrix} = A_{iso} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} + \begin{bmatrix} 0 & \alpha_{xy} & \alpha_{xz} \\ \alpha_{yx} & 0 & \alpha_{yz} \\ \alpha_{zx} & \alpha_{zy} & 0 \end{bmatrix} + \begin{bmatrix} \delta_{xx} & \delta_{xy} & \delta_{xz} \\ \delta_{yx} & \delta_{yy} & \delta_{yz} \\ \delta_{zx} & \delta_{zy} & \delta_{zz} \end{bmatrix} \quad (19\text{-}1)$$

$$\boldsymbol{A}_{2_{Cart}} = \boldsymbol{A}_0 + \boldsymbol{A}_1 + \boldsymbol{A}_2'$$

The latter formula uses a prime on the rank two component to avoid confusion with the original tensor. The various elements relate to each other through the following relationships[1].

$$A_{iso} = \frac{1}{3}Tr(\boldsymbol{A}_2) \qquad \alpha_{xy} = \frac{1}{2}[A_{xy} - A_{yx}] \qquad \delta_{xy} = \frac{1}{2}\left[A_{xy} + A_{yx} - \frac{2}{3}Tr(\boldsymbol{A}_2)\right] \tag{9-1}$$

The rank zero tensor is a scalar which will prove invariant under rotations of the coordinate system. The rank one tensor is usually zero because most spatial tensors are symmetric (or at least the elements α_{uv} are taken to be neglibible). Finally the rank 2 tensor will be traceless and symmetric, thus containing only five unique components[2]. Of the original nine components of $\boldsymbol{A_2}$ there remains nine unique components in this breakdown, 1, 3, and five for the respective tensors. Typically, one knows only about these tensor components when $\boldsymbol{A_2}$ is expressed relative to its principal axis system (PAS), that is, with respect to a set of axes where $\boldsymbol{A_2}$ appears diagonal. This situation appears as, from (19-1),

$$\boldsymbol{A}_2(PAS)_{Cart} = A_{iso}\begin{bmatrix}1&0&0\\0&1&0\\0&0&1\end{bmatrix} + \begin{bmatrix}0&\alpha_{xy}&\alpha_{xy}\\-\alpha_{xy}&0&\alpha_{yz}\\-\alpha_{xy}&-\alpha_{yz}&0\end{bmatrix} + \begin{bmatrix}\delta_{xx}&0&0\\0&\delta_{yy}&0\\0&0&\delta_{zz}\end{bmatrix} \tag{9-2}$$

When $\boldsymbol{A_2}$ is symmetric this is of course simply[3]

$$\boldsymbol{A}_2(PAS, sym)_{Cart} = A_{iso}\begin{bmatrix}1&0&0\\0&1&0\\0&0&1\end{bmatrix} + \begin{bmatrix}\delta_{xx}&0&0\\0&\delta_{yy}&0\\0&0&\delta_{zz}\end{bmatrix}. \tag{9-3}$$

Finally, since the rank 2 tensor is traceless it actually contains only two unique components. It is therefore more convenient to use an *anisotropy* value $\Delta\boldsymbol{A}$ and an *asymmetry* parameter, η, to de-

1. Equations (19-1) & (9-1) can easily be shown as consistent. The original tensor elements are reformed by
$$A_{xy} = \frac{1}{3}Tr\boldsymbol{A}_2 + \frac{1}{2}[A_{xy} - A_{yx}] + \frac{1}{2}\left(A_{xy} + A_{yx} - \frac{2}{3}Tr\boldsymbol{A}_2\right) = Tr(\boldsymbol{A}_2)\left[\frac{1}{3} - \frac{1}{3}\right] + A_{xy}\left[\frac{1}{2} + \frac{1}{2}\right] + A_{yx}\left[-\frac{1}{2} + \frac{1}{2}\right]$$
2. Being traceless, there are only two unique components on the diagonal of this array.
3. The matrix representation of the tensor $\boldsymbol{A_2}$ in the principal axis system will not be diagonal unless it is symmetric. Fortunately this is usually the case. For a non-symmetric matrix, it can also be broken up into the sum of a symmetric matrix and an anti-symmetric matrix which can be utilized in further calculations.

scribe the tensor[1].

$$\boldsymbol{A}_2(PAS, sym)_{Cart} = A_{iso}\boldsymbol{I} + \delta_{zz}\begin{bmatrix} -\frac{1}{2}(1-\eta) & 0 & 0 \\ 0 & -\frac{1}{2}(1+\eta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad \eta = (\delta_{xx} - \delta_{yy})/\delta_{zz} \qquad (9\text{-}4)$$

Prior to moving on to spherical tensors we show that the relationship $\Delta\boldsymbol{A} = \boldsymbol{3/2}\ \delta_{\boldsymbol{zz}}$ is valid.

$$\Delta A = A_{zz} - \frac{1}{2}(A_{xx} + A_{yy}) = (A_{iso} + \delta_{zz}) - \frac{1}{2}(A_{iso} + \delta_{xx} + A_{iso} + \delta_{yy}) = \delta_{zz} - \frac{1}{2}(\delta_{xx} + \delta_{yy})$$

$$\Delta A = \delta_{zz} + \frac{1}{2}\delta_{zz} - \frac{1}{2}(\delta_{xx} + \delta_{yy} + \delta_{zz}) = \frac{3}{2}\delta_{zz} - Tr\{A_2\} = \frac{3}{2}\delta_{zz}$$

1. The anisotropy value is a measure of the difference between the z-axis value and that in the xy-plane whereas the asymmetry parameter is a measure of the difference between the y-axis and the x-axis tensor values. The asymmetry parameter is related to δ_{zz} by $\Delta\boldsymbol{A} = \boldsymbol{3/2}\ \delta_{\boldsymbol{zz}}$.

9.8.3 Spherical Tensors

The irreducible spherical components of $\boldsymbol{A}$ are specified as $\boldsymbol{A_{l,m}}$. For an irreducible rank 1 spatial tensor the spherical and Cartesian components are inter-related by

$$A_{1,\pm1} = \mp\frac{1}{\sqrt{2}}[A_x \pm iA_y] = \mp\frac{1}{\sqrt{2}}A_{\pm}$$
$$A_{1,0} = A_z \qquad (9\text{-}5)$$
$$A_x = \frac{1}{\sqrt{2}}[-A_{1,1} + A_{1,-1}] \qquad A_y = \frac{i}{\sqrt{2}}[A_{1,1} + A_{1,-1}]$$

For a general rank 2 spatial tensor the irreducible spherical components can also be expressed in terms of the Cartesian components. In this case the following equations apply.

$$A_{0,0} = \frac{-1}{\sqrt{3}}[A_{xx} + A_{yy} + A_{zz}] = \frac{-1}{\sqrt{3}}Tr\{\boldsymbol{A}\}$$
$$A_{1,0} = \frac{-i}{\sqrt{2}}[A_{xy} - A_{yx}] \qquad A_{1,\pm1} = \frac{-1}{2}[(A_{zx} - A_{xz}) \pm i(A_{zy} - A_{yz})]$$
$$A_{2,0} = \frac{1}{\sqrt{6}}[-A_{xx} - A_{yy} + 2A_{zz}] = \frac{1}{\sqrt{6}}[3A_{zz} - Tr\{\boldsymbol{A}\}] \qquad (9\text{-}6)$$
$$A_{2,\pm1} = \mp\frac{1}{2}[(A_{xz} + A_{zx}) \pm i(A_{yz} + A_{zy})] \qquad A_{2,\pm2} = \frac{1}{2}[(A_{xx} - A_{yy}) \pm i(A_{xy} + A_{yx})]$$

The inverse equations, where the Cartesian components are expressed relative to the spherical components, are readily verifiable as the following.

$$A_{xx} = \frac{1}{2}(A_{2,2} + A_{2,-2}) - \frac{1}{\sqrt{6}}A_{2,0} - \frac{1}{\sqrt{3}}A_{0,0} \qquad A_{xy} = \frac{i}{\sqrt{2}}A_{1,0} - \frac{i}{2}(A_{2,2} - A_{2,-2})$$
$$A_{xz} = \frac{1}{2}[(A_{1,1} + A_{1,-1}) - (A_{2,1} - A_{2,-1})]$$
$$A_{yx} = -\frac{i}{\sqrt{2}}A_{1,0} - \frac{i}{2}(A_{2,2} - A_{2,-2}) \qquad A_{yy} = \frac{1}{2}(A_{2,2} + A_{2,-2}) + \frac{1}{\sqrt{6}}A_{2,0} + \frac{1}{\sqrt{3}}A_{0,0}$$
$$A_{yz} = \frac{i}{2}[-(A_{1,1} - A_{1,-1}) + (A_{2,1} + A_{2,-1})]$$
$$A_{zx} = -\frac{1}{2}[(A_{1,1} + A_{1,-1}) + (A_{2,1} - A_{2,-1})] \qquad A_{zy} = \frac{i}{2}[(A_{1,1} - A_{1,-1}) + (A_{2,1} + A_{2,-1})]$$
$$A_{zz} = \sqrt{\frac{2}{3}}A_{2,0} - \sqrt{\frac{1}{3}}A_{0,0}$$

A third set of useful equations relates the spherical components to the parameters $\boldsymbol{A_{iso}}$, δ_{zz}, and η

used to describe the symmetric rank two tensor. From equations (9-1) and (9-4),

$$\begin{gathered} A_{0,0} = -\sqrt{3}A_{iso} \\ A_{1,0} = -i\sqrt{2}\alpha_{xy} \qquad A_{1,\pm1} = \alpha_{xz} \pm i\alpha_{yz} \\ A_{2,0} = \sqrt{3/2}\delta_{zz} \qquad A_{2,\pm1} = 0 \qquad A_{2,\pm2} = \frac{1}{2}\delta_{zz}\mathrm{r} \end{gathered} \tag{9-7}$$

One can also formulate a Cartesian rank two tensor by taking the dyadic product of two vectors. A rank two spatial tensor produced from vectors ***u*** and ***v*** will be

$$\begin{bmatrix} u_x \\ u_y \\ u_z \end{bmatrix} \bullet \begin{bmatrix} v_x & v_y & v_z \end{bmatrix} = \begin{bmatrix} u_x v_x & u_x v_y & u_x v_z \\ u_y v_x & u_y v_y & u_y v_z \\ u_z v_x & u_z v_y & u_z v_z \end{bmatrix}. \tag{9-8}$$

The irreducible spherical components of ***A*** in this case are specified as ***A*$_{l,m}$*(u,v)***, the vector components (rank 1 tensors) which make up ***A*** specified in parenthesis. The nine formulas which relate the irreducible spherical components of ***A*** to the Cartesian components are the following.

$$\begin{gathered} A_{0,0}(u,v) = \frac{-1}{\sqrt{3}}\left[u_z v_z + \frac{1}{2}(u_+ v_- + u_- v_+)\right] \\ A_{1,0}(u,v) = \frac{-1}{2\sqrt{2}}[u_+ v_- + u_- v_+] \qquad A_{1,\pm1}(u,v) = \frac{-1}{2}[u_\pm v_z - u_z v_\pm] \\ A_{2,0}(u,v) = \frac{1}{\sqrt{6}}[3u_z v_z - (\boldsymbol{u} \bullet \boldsymbol{v})] \\ A_{2,\pm1}(u,v) = \mp\frac{1}{2}[u_\pm v_z + u_z v_\pm] \qquad A_{2,\pm2}(u,v) = \frac{1}{2}[u_\pm v_\pm] \end{gathered} \tag{9-9}$$

9.8.4 Tensor Rotations

The ***2l+1*** components of the irreducible spherical tensor T of rank ***l*** transform under a rotation of coordinate system (axes) according to[1]

$$T'_{l,m} = \boldsymbol{R}(\alpha,\beta,\gamma)T_{l,m}\boldsymbol{R}^{-1}(\alpha,\beta,\gamma) = \sum_{m'}^{\pm l} T_{l,m'}\boldsymbol{D}^{l}{}_{m',m}(\alpha,\beta,\gamma) \tag{9-10}$$

where $\boldsymbol{T'_{l,m}}$ are the tensor component with respect to the rotated axes and $\boldsymbol{D}^{l}{}_{m',m}$ the rank ***l*** Wigner rotation matrix elements. When a rotation of the coordinate system is performed the tensor component $\boldsymbol{T_{l,m}}$ is transformed into a linear combination of ***2l+1*** components of the original tensor.

As a relatively simple example, we consider the rotation of a rank 1 irreducible spherical tensor-equivalent to the rotation of a vector. Three simple spatial vectors are the unit vectors along the coordinate axes. Their Cartesian and analogous Spherical Tensor representations are given in equation (9-11).

$$\begin{matrix}\boldsymbol{u}\\ \begin{bmatrix}u_x\\u_y\\u_z\end{bmatrix}\end{matrix} \Leftrightarrow \begin{matrix}\boldsymbol{x}\\ \begin{bmatrix}1\\0\\0\end{bmatrix}\end{matrix}, \begin{matrix}\boldsymbol{y}\\ \begin{bmatrix}0\\1\\0\end{bmatrix}\end{matrix}, \begin{matrix}\boldsymbol{z}\\ \begin{bmatrix}0\\0\\1\end{bmatrix}\end{matrix} \qquad \begin{matrix}\boldsymbol{A}\\ \begin{bmatrix}-\frac{1}{\sqrt{2}}(x+iy)\\ z \\ \frac{1}{\sqrt{2}}(x-iy)\end{bmatrix}\end{matrix} \Leftrightarrow \begin{matrix}\boldsymbol{x}\\ \frac{1}{\sqrt{2}}\begin{bmatrix}-1\\0\\1\end{bmatrix}\end{matrix}, \begin{matrix}\boldsymbol{y}\\ \frac{1}{\sqrt{2}}\begin{bmatrix}-i\\0\\-i\end{bmatrix}\end{matrix}, \begin{matrix}\boldsymbol{z}\\ \begin{bmatrix}0\\1\\0\end{bmatrix}\end{matrix} \tag{9-11}$$

According to (9-10), the components of a rank 1 spatial tensor relative to the rotated coordinate system will be

$$A'_{1,m} = \sum_{m'}^{\pm 1} A_{1,m'}\boldsymbol{D}^{1}{}_{m',m}(\Omega) = A_{1,1}\boldsymbol{D}^{1}{}_{1,m}(\Omega) + A_{1,0}\boldsymbol{D}^{1}{}_{0,m}(\Omega) + A_{1,-1}\boldsymbol{D}^{1}{}_{-1,m}(\Omega). \tag{9-12}$$

Here Ω is used as shorthand notation for the set of Euler angles α, β, and γ. The reduced Wigner rotation matrix elements can be found in sosi, sosi in the chapter Spatial Functions and relate to the $\boldsymbol{D}^{1}{}_{m',m}$ in the previous equation by the simple formula sosi.

The following equation describes the unit z-vector under a rotation of its coordinate system about

1. Brink and Satchler, page 53, equation (4.8). As these authors point out, the rotation defined here is a rotation of axes and this is the inverse of a rotation on the system. Equations involving rotations of the system will appear similar to (9-10) but the positions of $\boldsymbol{R}$ and $\boldsymbol{R}^{-1}$ will be switched.

the y-axis by ninety degrees. In this case the Euler angles are (0, 90, 0).

$$\boldsymbol{A} = \boldsymbol{z} \qquad \boldsymbol{A}' \qquad \boldsymbol{A}'$$

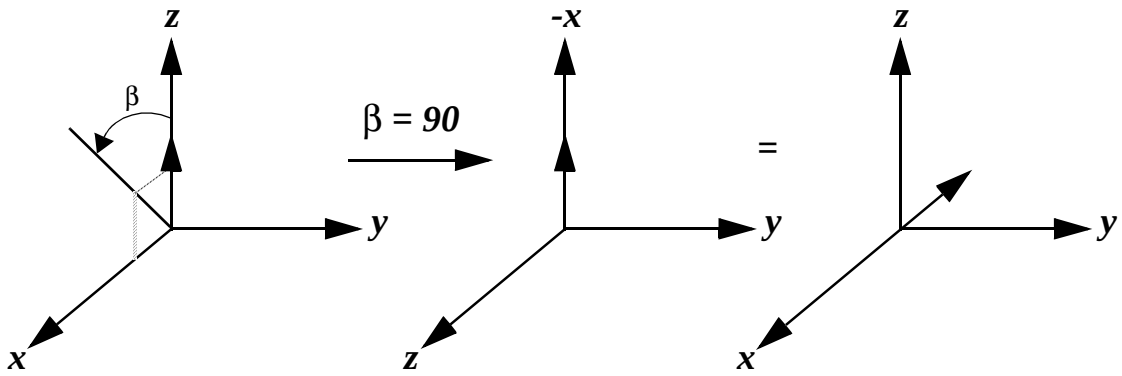
$$\begin{bmatrix}0\\1\\0\end{bmatrix} \rightarrow \begin{bmatrix} A_{1,1}\boldsymbol{D}^1{}_{1,1}(\Omega) + A_{1,0}\boldsymbol{D}^1{}_{0,1}(\Omega) + A_{1,-1}\boldsymbol{D}^1{}_{-1,1}(\Omega) \\ A_{1,1}\boldsymbol{D}^1{}_{1,0}(\Omega) + A_{1,0}\boldsymbol{D}^1{}_{0,1}(\Omega) + A_{1,-1}\boldsymbol{D}^1{}_{-1,0}(\Omega) \\ A_{1,1}\boldsymbol{D}^1{}_{1,-1}(\Omega) + A_{1,0}\boldsymbol{D}^1{}_{0,1}(\Omega) + A_{1,-1}\boldsymbol{D}^1{}_{-1,-1}(\Omega) \end{bmatrix} = \begin{bmatrix} A_{1,0}\boldsymbol{D}^1{}_{0,1}(\Omega) \\ A_{1,0}\boldsymbol{D}^1{}_{0,1}(\Omega) \\ A_{1,0}\boldsymbol{D}^1{}_{0,1}(\Omega) \end{bmatrix} . \quad (9\text{-}13)$$

Utilization of the formula and table previous mentioned produces

$$\boldsymbol{A}' = \begin{bmatrix} A_{1,0}\boldsymbol{d}^1{}_{0,1}(90) \\ A_{1,0}\boldsymbol{d}^1{}_{0,1}(90) \\ A_{1,0}\boldsymbol{d}^1{}_{0,1}(90) \end{bmatrix} = \frac{1}{\sqrt{2}}\begin{bmatrix}1\\0\\-1\end{bmatrix} = -\boldsymbol{x},$$

which can be seen from (9-11) as the spherical tensor representation of a unit vector along the negative x-axis. This can be also depicted graphically with the following diagram. Keep in mind the rotations are right-handed and here it is the axes which rotate, not the system.

Coordinate System Rotation about Euler Angles (α=0, β=90, γ=0)

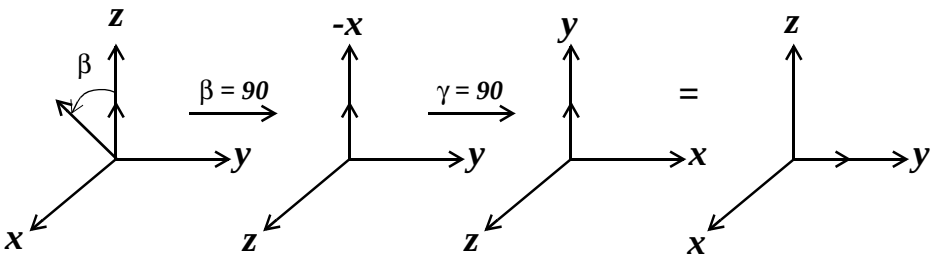


Rotating the coordinate system about the x-axis uses the Euler angles (0, 90, 90) and the following applies.

$$\boldsymbol{A}' = \begin{bmatrix} A_{1,0}\boldsymbol{d}^1{}_{0,1}(90)exp(-i90) \\ A_{1,0}\boldsymbol{d}^1{}_{0,1}(90)exp(0) \\ A_{1,0}\boldsymbol{d}^1{}_{0,1}(90)exp(i90) \end{bmatrix} = \frac{1}{\sqrt{2}}\begin{bmatrix}-i\\0\\-i\end{bmatrix}$$

This time the tensor corresponds to the unit vector along the y-axis. The graphical representation is below.

Coordinate System Rotation about Euler Angles (α=90, β=90, γ=0)



Rotations on vectors (rank 1 tensors) can of course be done entirely with respect to the Cartesian axes. It may then seem that use of spherical tensors is unnecessary and too complicated. For higher rank tensors this is not true. For tensors of rank 2 or more it is far easier to preform rotations in terms of irreducible spherical tensors.

9.8.5 Tensor Products

The product of two **irreducible** spherical tensors produces another spherical tensor, ***T = T1 x T2***, whose irreducible components are given by[1]

$$T_{L,M} = (-1)^{2l1+L-M}\sqrt{2L+1}\sum_{m1\,m2}^{\pm l1\pm l2}\sum T1_{l1,m1}T2_{l2,m2}\begin{pmatrix} L & l1 & l2 \\ M & -m1 & -m2 \end{pmatrix}. \tag{9-14}$$

where $L \in [l1+l2, |l1-l2|]$, $M = m1 + m2$ and

$$\begin{pmatrix} L & l1 & l2 \\ M & -m1 & -m2 \end{pmatrix}$$

are the Wigner 3-j coefficients. This formula can also be expressed in terms of Clebsch-Gordan coefficients.

$$A_{L,M} = \sum_{m1\,m2}^{\pm l1\pm l2}\sum A1_{l1,m1}A2_{l2,m2}\langle l1\,l2\,m1\,m2|LM\rangle. \tag{9-15}$$

As depicted in the following diagram, the two irreducible tensors combine to form a reducible ten-

1. This equation corresponds to Brink and Satchler, page 123 in Appendix VI. For more discussion of tensor products see Chapter “Spin Tensor” of this document.

sor with rank equal to the sum of the ranks of two tensor which formed it.

IRREDUCIBLE $\mathbf{T}_l$: $\mathbf{T}_{lm}$		REDUCIBLE	IRREDUCIBLE $\mathbf{T}_L$: $\mathbf{T}_{LM}$
$\mathbf{T}_0$: $\mathbf{T}_{00}$ $\mathbf{T}_0$: $\mathbf{T}_{00}$	x →		$\mathbf{T}_0$: $\mathbf{T}_{00}$
$\mathbf{T}_0$: $\mathbf{T}_{00}$ $\mathbf{T}_1$: $\mathbf{T}_{11}$ $\mathbf{T}_{10}$ $\mathbf{T}_{1-1}$	x →		$\mathbf{T}_0$: None $\mathbf{T}_1$: $\mathbf{T}_{11}$ $\mathbf{T}_{10}$ $\mathbf{T}_{1-1}$
$\mathbf{T}_1$: $\mathbf{T}_{11}$ $\mathbf{T}_{10}$ $\mathbf{T}_{1-1}$ $\mathbf{T}_1$: $\mathbf{T}_{11}$ $\mathbf{T}_{10}$ $\mathbf{T}_{1-1}$	x →	$\mathbf{T}_2$	$\mathbf{T}_0$: $\mathbf{T}_{00}$ $\mathbf{T}_1$: $\mathbf{T}_{11}$ $\mathbf{T}_{10}$ $\mathbf{T}_{1-1}$ $\mathbf{T}_2$: $\mathbf{T}_{22}$ $\mathbf{T}_{21}$ $\mathbf{T}_{20}$ $\mathbf{T}_{2-1}$ $\mathbf{T}_{2-2}$
$\mathbf{T}_1$: $\mathbf{T}_{11}$ $\mathbf{T}_{10}$ $\mathbf{T}_{1-1}$ $\mathbf{T}_2$: $\mathbf{T}_{22}$ $\mathbf{T}_{21}$ $\mathbf{T}_{20}$ $\mathbf{T}_{2-1}$ $\mathbf{T}_{2-2}$	x →	$\mathbf{T}_3$	$\mathbf{T}_0$: None $\mathbf{T}_1$: $\mathbf{T}_{11}$ $\mathbf{T}_{10}$ $\mathbf{T}_{1-1}$ $\mathbf{T}_2$: $\mathbf{T}_{22}$ $\mathbf{T}_{21}$ $\mathbf{T}_{20}$ $\mathbf{T}_{2-1}$ $\mathbf{T}_{2-2}$ $\mathbf{T}_3$: $\mathbf{T}_{33}$ $\mathbf{T}_{32}$ $\mathbf{T}_{31}$ $\mathbf{T}_{30}$ $\mathbf{T}_{3-1}$ $\mathbf{T}_{3-2}$ $\mathbf{T}_{3-3}$

Figure B : Irreducible components resulting from Tensor Products.

As shown, the tensor formed from the product is reducible and can be broken up into irreducible tensors of its rank down to rank zero. The reducible tensor resulting from the multiplication is stored internally as irreducible tensors whose components are in the right hand column. A more complicated situation occurs when multiplying two reducible tensors. In that instance a sum is performed over all the irreducible components. The following demonstrates the multiplication of a reducible rank 2 tensor with an irreducible rank 1 tensor to form a reducible rank 3 tensor.

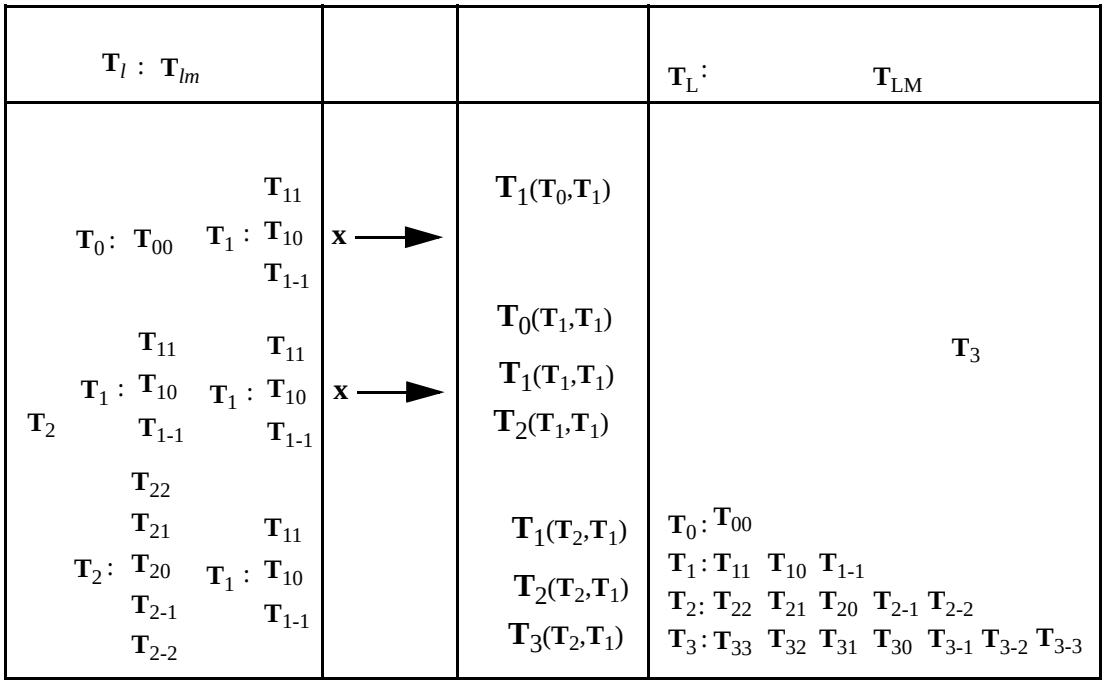


Figure C : Irreducible components resulting from Tensor Products.

To demonstrate the mathematics we shall first consider the buildup of a rank 2 spatial tensor from the product of two irreducible rank 1 spatial tensors whose components are the normalized spherical harmonics. According to our formula this would be, switching from uppercase T to uppercase A to indicate a spatial tensor,

$$A_{L,M}(1,2) = (-1)^{2+L-M}\sqrt{2L+1}\sum_{m1}^{\pm1}\sum_{m2}^{\pm1} A1_{1,m1}A2_{1,m2}\begin{pmatrix} L & 1 & 1 \\ M & -m1 & -m2 \end{pmatrix} \tag{9-16}$$

The components of the first rank 1 tensor for this example are the normalized spherical harmonics (See equation sosi sosi.)

$$A1_{1,0} = Y_{1,0}(1) = \sqrt{\frac{3}{4\pi}}\cos\theta \qquad A1_{1,\pm1} = Y_{1,\pm1}(1) = \mp\sqrt{\frac{3}{8\pi}}\sin\theta e^{\pm i\phi}, \tag{9-17}$$

and the components of the second rank 1 spatial tensor the same with the index switched from 1 to 2. Proceeding to build up the l = 2 components of the resultant rank 2 tensor, we have from equation (9-16)

$$A_{2,M}(1,2) = (-1)^{2+2-M}\sqrt{2(2)+1}\sum_{m1}^{\pm1}\sum_{m2}^{\pm1} A1_{1,m1}A2_{1,m2}\begin{pmatrix} 2 & 1 & 1 \\ M & -m1 & -m2 \end{pmatrix}.$$

$$A_{2,M}(1,2) = (-1)^{M}\sqrt{5}\sum_{m1}^{\pm1}\sum_{m2}^{\pm1} Y_{1,m1}(1)Y_{1,m2}(2)\begin{pmatrix} 2 & 1 & 1 \\ M & -m1 & -m2 \end{pmatrix} \tag{9-18}$$

This formula applies to five components, M = 0, 1, -1, 2 and -2.

$$A_{2,0}(1,2) = \sqrt{5}\left[Y_{1,0}(1)Y_{1,0}(2)\begin{pmatrix} 2 & 1 & 1 \\ 0 & 0 & 0 \end{pmatrix} + Y_{1,1}(1)Y_{1,-1}(2)\begin{pmatrix} 2 & 1 & 1 \\ 0 & -1 & 1 \end{pmatrix} + Y_{1,-1}(1)Y_{1,1}(2)\begin{pmatrix} 2 & 1 & 1 \\ 0 & 1 & -1 \end{pmatrix}\right]$$

$$A_{2,\pm1}(1,2) = -\sqrt{5}\left[Y_{1,0}(1)Y_{1,\pm1}(2)\begin{pmatrix} 2 & 1 & 1 \\ 1 & 0 & -1 \end{pmatrix} + Y_{1,\pm1}(1)Y_{1,0}(2)\begin{pmatrix} 2 & 1 & 1 \\ 1 & -1 & 0 \end{pmatrix}\right]$$

$$A_{2,\pm2}(1,2) = \sqrt{5}\left[Y_{1,\pm1}(1)Y_{1,\pm1}(2)\begin{pmatrix} 2 & 1 & 1 \\ 2 & -1 & -1 \end{pmatrix}\right]$$

Using the fact that an interchange of two adjacent columns multiplies the coefficient by the factor $(-1)^{a+b+c}$ as well as the identity

$$\begin{pmatrix} a & b & c \\ -\alpha & -\beta & -\gamma \end{pmatrix} = (-1)^{a+b+c}\begin{pmatrix} a & b & c \\ \alpha & \beta & \gamma \end{pmatrix}, \tag{9-19}$$

we obtain

$$A_{2,0}(1,2) = \sqrt{5}\left[Y_{1,0}(1)Y_{1,0}(2)\begin{pmatrix} 2 & 1 & 1 \\ 0 & 0 & 0 \end{pmatrix} + \begin{pmatrix} 2 & 1 & 1 \\ 0 & -1 & 1 \end{pmatrix}[Y_{1,1}(1)Y_{1,-1}(2) + Y_{1,-1}(1)Y_{1,1}(2)]\right]$$

$$A_{2,1}(1,2) = -\sqrt{5}\begin{pmatrix} 2 & 1 & 1 \\ 1 & 0 & -1 \end{pmatrix}[Y_{1,0}(1)Y_{1,1}(2) + Y_{1,1}(1)Y_{1,0}(2)]$$

$$A_{2,-1}(1,2) = -\sqrt{5}\begin{pmatrix} 2 & 1 & 1 \\ 1 & 0 & -1 \end{pmatrix}[Y_{1,0}(1)Y_{1,-1}(2) + Y_{1,-1}(1)Y_{1,0}(2)]$$

$$A_{2,2}(1,2) = \sqrt{5}\left[Y_{1,1}(1)Y_{1,1}(2)\begin{pmatrix} 2 & 1 & 1 \\ 2 & -1 & -1 \end{pmatrix}\right] \qquad A_{2,-2}(1,2) = \sqrt{5}\left[Y_{1,-1}(1)Y_{1,-1}(2)\begin{pmatrix} 2 & 1 & 1 \\ 2 & -1 & -1 \end{pmatrix}\right]$$

The four needed Wigner coefficients are

$$\begin{pmatrix} 2 & 1 & 1 \\ 0 & 0 & 0 \end{pmatrix} = \frac{2}{\sqrt{30}} \qquad \begin{pmatrix} 2 & 1 & 1 \\ 0 & -1 & 1 \end{pmatrix} = \frac{1}{\sqrt{30}} \qquad \begin{pmatrix} 2 & 1 & 1 \\ 1 & 0 & -1 \end{pmatrix} = -\sqrt{\frac{1}{10}} \qquad \begin{pmatrix} 2 & 1 & 1 \\ 2 & -1 & -1 \end{pmatrix} = \sqrt{\frac{1}{5}}$$

so the rank 2 tensor components become

$$A_{2,0}(1,2) = \sqrt{\frac{1}{6}}[2Y_{1,0}(1)Y_{1,0}(2) + [Y_{1,1}(1)Y_{1,-1}(2) + Y_{1,-1}(1)Y_{1,1}(2)]]$$

$$A_{2,\pm1}(1,2) = \sqrt{\frac{1}{2}}[Y_{1,0}(1)Y_{1,\pm1}(2) + Y_{1,\pm1}(1)Y_{1,0}(2)] \qquad A_{2,\pm2}(1,2) = Y_{1,\pm1}(1)Y_{1,\pm1}(2)$$

Plugging in the rank 1 normalized spherical harmonics[1] produces the following equations.

1. See equation sosi in the Spatial Functions chapter.

$$A_{2,0}(1,2) = \sqrt{\frac{1}{6}}\left[\frac{6}{4\pi}\cos\theta_1\cos\theta_2 - \frac{3}{8\pi}\sin\theta_1\sin\theta_2(e^{i\phi_1}e^{-i\phi_2} + e^{-i\phi_1}e^{i\phi_2})\right]$$

$$A_{2,\pm1}(1,2) = \sqrt{\frac{1}{2}}\left[\mp\frac{3}{4\pi\sqrt{2}}(\cos\theta_1\sin\theta_2 e^{\pm i\phi_2} + \sin\theta_1 e^{\pm i\phi_1}\cos\theta_2)\right]$$

$$A_{2,\pm2}(1,2) = \frac{3}{8\pi}\sin\theta_1 e^{\pm i\phi_1}\sin\theta_2 e^{\pm i2}$$

When both angles are equivalent, i.e. 1 = 2, it is immediately seen that the rank 2 components of the resulting tensor are directly related to the rank 2 normalized spherical harmonics[1].

$$A_{2,0}(\theta,\phi) = \sqrt{\frac{3}{32\pi^2}}(2\cos^2\theta - \sin^2\theta) = \sqrt{\frac{3}{32\pi^2}}(3\cos^2\theta - 1) = \sqrt{\frac{3}{10\pi}}Y_{2,0}(\theta,\phi)$$

$$A_{2,\pm1}(\theta,\phi) = \mp\frac{6}{8\pi}[\cos\theta\sin\theta e^{\pm i\phi}] = \sqrt{\frac{3}{10\pi}}Y_{2,\pm1}(\theta,\phi)$$

$$A_{2,\pm2}(\theta,\phi) = \frac{3}{8\pi}\sin^2\theta e^{\pm 2i\phi} = \sqrt{\frac{3}{10\pi}}Y_{2,\pm2}(\theta,\phi)$$

Evidently, in this case we produce a tensor with components being the normalized spherical harmonics scaled by the factor $\sqrt{3/(10\pi)}$,

$$A_{2,m}(\theta,\phi) = \sqrt{\frac{3}{10\pi}}Y_{2,m}(\theta,\phi) \ .$$

Keep in mind that this results from the specific build up of a rank 2 spatial tensor from two spherical irreducible rank1 spatial tensors whose components were the normalized spherical harmonics. Since a spherical irreducible tensor can be multiplied by any scalar and still remain spherical and irreducible, i.e. multiplication of all tensor components by the same number does not change the tensor properties, a rank two tensor with normalized spherical harmonics will also be a valid spherical tensor. In fact we shall continue to build up the rank 3 l=3 tensor components from the product of the rank 1 and rank 2 tensors both containing normalized spherical harmonics. Analogous to (9-16) we have

$$A_{L,M}(1,2) = (-1)^{2+L-M}\sqrt{2L+1}\sum_{m1,m2}^{\pm1\ \pm2}\sum A1_{1,m1}A2_{2,m2}\begin{pmatrix} L & 1 & 2 \\ M & -m1 & -m2 \end{pmatrix},$$

and we specifically desire

$$A_{3,M}(1,2) = (-1)^{2+3-M}\sqrt{2(3)+1}\sum_{m1,m2}^{\pm1\ \pm2}\sum A1_{1,m1}A2_{2,m2}\begin{pmatrix} 3 & 1 & 2 \\ M & -m1 & -m2 \end{pmatrix}$$

or

1. Also see equation sosi in the Spatial Functions chapter.

$$A_{3,M}(1,2) = (-1)^{5-M}\sqrt{7}\sum_{m1}^{+1}\sum_{m2}^{+2} A1_{1,m1}A2_{2,m2}\begin{pmatrix} 3 & 1 & 2 \\ M & -m1 & -m2 \end{pmatrix}.$$

Since we wish to use the normalized spherical harmonics, we have the following 7 equations, corresponding to the components M = 0, 1, -1, 2,-2, 3 and -3.

$$A_{3,0}(1,2) = -\sqrt{7}\left[Y_{1,0}(1)Y_{2,0}(2)\begin{pmatrix} 3 & 1 & 2 \\ 0 & 0 & 0 \end{pmatrix} + Y_{1,1}(1)Y_{2,-1}(2)\begin{pmatrix} 3 & 1 & 2 \\ 0 & -1 & 1 \end{pmatrix} + Y_{1,-1}(1)Y_{2,1}(2)\begin{pmatrix} 3 & 1 & 2 \\ 0 & 1 & -1 \end{pmatrix}\right]$$

$$A_{3,1}(1,2) = \sqrt{7}\left[Y_{1,0}(1)Y_{2,1}(2)\begin{pmatrix} 3 & 1 & 2 \\ 1 & 0 & -1 \end{pmatrix} + Y_{1,1}(1)Y_{2,0}(2)\begin{pmatrix} 3 & 1 & 2 \\ 1 & -1 & 0 \end{pmatrix} + Y_{1,-1}(1)Y_{2,2}(2)\begin{pmatrix} 3 & 1 & 2 \\ 1 & 1 & -2 \end{pmatrix}\right]$$

$$A_{3,-1}(1,2) = \sqrt{7}\left[Y_{1,0}(1)Y_{2,-1}(2)\begin{pmatrix} 3 & 1 & 2 \\ -1 & 0 & 1 \end{pmatrix} + Y_{1,-1}(1)Y_{2,0}(2)\begin{pmatrix} 3 & 1 & 2 \\ -1 & 1 & 0 \end{pmatrix} + Y_{1,1}(1)Y_{2,-2}(2)\begin{pmatrix} 3 & 1 & 2 \\ -1 & -1 & 2 \end{pmatrix}\right]$$

$$A_{3,2}(1,2) = -\sqrt{7}\left[Y_{1,1}(1)Y_{2,1}(2)\begin{pmatrix} 3 & 1 & 2 \\ 2 & -1 & -1 \end{pmatrix} + Y_{1,0}(1)Y_{2,2}(2)\begin{pmatrix} 3 & 1 & 2 \\ 2 & 0 & -2 \end{pmatrix}\right]$$

$$A_{3,-2}(1,2) = -\sqrt{7}\left[Y_{1,-1}(1)Y_{2,-1}(2)\begin{pmatrix} 3 & 1 & 2 \\ -2 & 1 & 1 \end{pmatrix} + Y_{1,0}(1)Y_{2,-2}(2)\begin{pmatrix} 3 & 1 & 2 \\ -2 & 0 & 2 \end{pmatrix}\right]$$

$$A_{3,3}(1,2) = \sqrt{7}\left[Y_{1,1}(1)Y_{2,2}(2)\begin{pmatrix} 3 & 1 & 2 \\ 3 & -1 & -2 \end{pmatrix}\right] \qquad A_{3,-2}(1,2) = \sqrt{7}\left[Y_{1,-1}(1)Y_{2,-2}(2)\begin{pmatrix} 3 & 1 & 2 \\ -3 & 1 & 2 \end{pmatrix}\right]$$

Again using the fact that an interchange of two adjacent columns multiplies the coefficient by the factor $(-1)^{a+b+c}$ as well as the identity[1]

$$\begin{pmatrix} a & b & c \\ -\alpha & -\beta & -\gamma \end{pmatrix} = (-1)^{a+b+c}\begin{pmatrix} a & b & c \\ \alpha & \beta & \gamma \end{pmatrix}, \tag{9-20}$$

we obtain

1. Brink and Satchler, Appendix 1, Page 113.

$$A_{3,0}(1,2) = -\sqrt{7}\left[Y_{1,0}(1)Y_{2,0}(2)\begin{pmatrix} 3 & 1 & 2 \\ 0 & 0 & 0 \end{pmatrix} + [Y_{1,1}(1)Y_{2,-1}(2) + Y_{1,-1}(1)Y_{2,1}(2)]\begin{pmatrix} 3 & 1 & 2 \\ 0 & -1 & 1 \end{pmatrix}\right]$$

$$A_{3,1}(1,2) = \sqrt{7}\left[Y_{1,0}(1)Y_{2,1}(2)\begin{pmatrix} 3 & 1 & 2 \\ 1 & 0 & -1 \end{pmatrix} + Y_{1,1}(1)Y_{2,0}(2)\begin{pmatrix} 3 & 1 & 2 \\ 1 & -1 & 0 \end{pmatrix} + Y_{1,-1}(1)Y_{2,2}(2)\begin{pmatrix} 3 & 1 & 2 \\ 1 & 1 & -2 \end{pmatrix}\right]$$

$$A_{3,-1}(1,2) = \sqrt{7}\left[Y_{1,0}(1)Y_{2,-1}(2)\begin{pmatrix} 3 & 1 & 2 \\ 1 & 0 & -1 \end{pmatrix} + Y_{1,-1}(1)Y_{2,0}(2)\begin{pmatrix} 3 & 1 & 2 \\ 1 & -1 & 0 \end{pmatrix} + Y_{1,1}(1)Y_{2,-2}(2)\begin{pmatrix} 3 & 1 & 2 \\ 1 & 1 & -2 \end{pmatrix}\right]$$

$$A_{3,2}(1,2) = -\sqrt{7}\left[Y_{1,1}(1)Y_{2,1}(2)\begin{pmatrix} 3 & 1 & 2 \\ -2 & 1 & 1 \end{pmatrix} + Y_{1,0}(1)Y_{2,2}(2)\begin{pmatrix} 3 & 1 & 2 \\ 2 & 0 & -2 \end{pmatrix}\right]$$

$$A_{3,-2}(1,2) = -\sqrt{7}\left[Y_{1,-1}(1)Y_{2,-1}(2)\begin{pmatrix} 3 & 1 & 2 \\ -2 & 1 & 1 \end{pmatrix} + Y_{1,0}(1)Y_{2,-2}(2)\begin{pmatrix} 3 & 1 & 2 \\ 2 & 0 & -2 \end{pmatrix}\right]$$

$$A_{3,3}(1,2) = \sqrt{7}\left[Y_{1,1}(1)Y_{2,2}(2)\begin{pmatrix} 3 & 1 & 2 \\ 3 & -1 & -2 \end{pmatrix}\right] \qquad A_{3,-3}(1,2) = \sqrt{7}\left[Y_{1,-1}(1)Y_{2,-2}(2)\begin{pmatrix} 3 & 1 & 2 \\ 3 & -1 & -2 \end{pmatrix}\right]$$

The eight needed Wigner coefficients are determined in the following way[1].

$$\begin{pmatrix} a & a+2 & 2 \\ \alpha & -\alpha & 0 \end{pmatrix} = (-1)^{a-\alpha}\sqrt{\frac{3(a+\alpha+1)(a+\alpha+2)(a-\alpha+1)(a-\alpha+2)}{(a+1)(2a+5)(2a+4)(2a+3)(2a+1)}} \rightarrow \begin{pmatrix} 1 & 3 & 2 \\ 0 & 0 & 0 \end{pmatrix} = \begin{pmatrix} 3 & 1 & 2 \\ 0 & 0 & 0 \end{pmatrix} = -\sqrt{\frac{3}{35}}$$

$$\begin{pmatrix} a & a+2 & 2 \\ \alpha-\alpha-1 & 1 \end{pmatrix} = (-1)^{a-\alpha-1}\sqrt{\frac{(a+\alpha+1)(a+\alpha+2)(a+\alpha+3)(a-\alpha+1)}{(a+1)(a+2)(2a+1)(2a+3)(2a+5)}} \rightarrow \begin{pmatrix} 1 & 3 & 2 \\ -1 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 3 & 1 & 2 \\ 0 & -1 & 1 \end{pmatrix} = -\sqrt{\frac{1}{35}}$$

$$\begin{pmatrix} a & a+2 & 2 \\ \alpha-\alpha-1 & 1 \end{pmatrix} = (-1)^{a-\alpha-1}\sqrt{\frac{(a+\alpha+1)(a+\alpha+2)(a+\alpha+3)(a-\alpha+1)}{(a+1)(a+2)(2a+1)(2a+3)(2a+5)}} \rightarrow \begin{pmatrix} 1 & 3 & 2 \\ 0 & -1 & 1 \end{pmatrix} = \begin{pmatrix} 3 & 1 & 2 \\ 1 & 0 & -1 \end{pmatrix} = \sqrt{\frac{8}{105}}$$

$$\begin{pmatrix} a & a+2 & 2 \\ \alpha & -\alpha & 0 \end{pmatrix} = (-1)^{a-\alpha}\sqrt{\frac{3(a+\alpha+1)(a+\alpha+2)(a-\alpha+1)(a-\alpha+2)}{(a+1)(2a+5)(2a+4)(2a+3)(2a+1)}} \rightarrow \begin{pmatrix} 1 & 3 & 2 \\ 1 & -1 & 0 \end{pmatrix} = \begin{pmatrix} 3 & 1 & 2 \\ 1 & -1 & 0 \end{pmatrix} = \sqrt{\frac{2}{35}}$$

$$\begin{pmatrix} a & a+2 & 2 \\ \alpha-\alpha-2 & 2 \end{pmatrix} = (-1)^{a-\alpha}\sqrt{\frac{(a+\alpha+1)(a+\alpha+2)(a+\alpha+3)(a+\alpha+4)}{(2a+5)(2a+4)(2a+3)(2a+2)(2a+1)}} \rightarrow \begin{pmatrix} 1 & 3 & 2 \\ -1 & -1 & 2 \end{pmatrix} = \begin{pmatrix} 3 & 1 & 2 \\ 1 & 1 & -2 \end{pmatrix} = -\sqrt{\frac{1}{105}}$$

$$\begin{pmatrix} a & a+2 & 2 \\ \alpha-\alpha-1 & 1 \end{pmatrix} = (-1)^{a-\alpha-1}\sqrt{\frac{(a+\alpha+1)(a+\alpha+2)(a+\alpha+3)(a-\alpha+1)}{(a+1)(a+2)(2a+1)(2a+3)(2a+5)}} \rightarrow \begin{pmatrix} 1 & 3 & 2 \\ 1 & -2 & 1 \end{pmatrix} = \begin{pmatrix} 3 & 1 & 2 \\ -2 & 1 & 1 \end{pmatrix} = -\sqrt{\frac{2}{21}}$$

$$\begin{pmatrix} a & a+2 & 2 \\ \alpha-\alpha-2 & 2 \end{pmatrix} = (-1)^{a-\alpha}\sqrt{\frac{(a+\alpha+1)(a+\alpha+2)(a+\alpha+3)(a+\alpha+4)}{(2a+5)(2a+4)(2a+3)(2a+2)(2a+1)}} \rightarrow \begin{pmatrix} 1 & 3 & 2 \\ 0 & -2 & 2 \end{pmatrix} = \begin{pmatrix} 3 & 1 & 2 \\ 2 & 0 & -2 \end{pmatrix} = -\sqrt{\frac{1}{21}}$$

$$\begin{pmatrix} a & a+2 & 2 \\ \alpha-\alpha-2 & 2 \end{pmatrix} = (-1)^{a-\alpha}\sqrt{\frac{(a+\alpha+1)(a+\alpha+2)(a+\alpha+3)(a+\alpha+4)}{(2a+5)(2a+4)(2a+3)(2a+2)(2a+1)}} \rightarrow \begin{pmatrix} 1 & 3 & 2 \\ 1 & -3 & 2 \end{pmatrix} = \begin{pmatrix} 3 & 1 & 2 \\ 3 & -1 & -2 \end{pmatrix} = \sqrt{\frac{1}{7}}$$

The equations describing the rank 3 tensor components are then

1. These formulae are taken from Brink and Satchler, TABLE 3, page 36.

$$A_{3,0}(1,2) = \frac{1}{\sqrt{5}}[3Y_{1,0}(1)Y_{2,0}(2) + \{Y_{1,1}(1)Y_{2,-1}(2) + Y_{1,-1}(1)Y_{2,1}(2)\}]$$

$$A_{3,\pm1}(1,2) = \left[Y_{1,0}(1)Y_{2,\pm1}(2)\left(\sqrt{\frac{8}{15}}\right) + Y_{1,\pm1}(1)Y_{2,0}(2)\left(\sqrt{\frac{2}{5}}\right) - Y_{1,\mp1}(1)Y_{2,\pm2}(2)\left(\sqrt{\frac{1}{15}}\right)\right]$$

$$A_{3,\pm2}(1,2) = \sqrt{\frac{1}{3}}[\sqrt{2}Y_{1,\pm1}(1)Y_{2,\pm1}(2) + Y_{1,0}(1)Y_{2,\pm2}(2)]$$

$$A_{3,\pm3}(1,2) = [Y_{1,\pm1}(1)Y_{2,\pm2}(2)]$$

$$A_{3,0}(1,2) = \sqrt{\frac{9}{64\pi^2}}[\cos\theta_1(3\cos^2\theta_2 - 1) - \sin\theta_1\cos\theta_2\sin\theta_2[e^{i\phi_1}e^{-i\phi_2} + e^{-i\phi_1}e^{i\phi_2}]$$

$$A_{3,\pm1}(1,2) = \mp\sqrt{\frac{3}{4\pi^2}}\left[\cos\theta_1(\cos\theta_2\sin\theta_2 e^{\pm i\phi_2}) + \sqrt{\frac{1}{16}}(\sin\theta_1 e^{\pm i\phi_1})(3\cos^2\theta_2 - 1) + \sqrt{\frac{1}{64}}(\sin\theta_1 e^{\mp i\phi_1})(\sin^2\theta_2 e^{\pm 2i\phi_2})\right]$$

$$A_{3,\pm2}(1,2) = \sqrt{\frac{15}{64\pi^2}}\left[\sqrt{2}(\sin\theta_1 e^{\pm i\phi_1})(\cos\theta_2\sin\theta_2 e^{\pm i\phi_2}) + \sqrt{\frac{1}{2}}\cos\theta_1(\sin^2\theta_2 e^{\pm 2i\phi_2})\right]$$

$$A_{3,\pm3}(1,2) = \mp\sqrt{\frac{45}{256\pi^2}}(\sin\theta_1 e^{\pm i\phi_1}\sin^2\theta_2 e^{\pm 2i\phi_2})$$

and when both angles are the same,

$$A_{3,0}(\theta,\phi) = \sqrt{\frac{9}{64\pi^2}}[\cos\theta(3\cos^2\theta - 1) - 2\cos\theta\sin^2\theta = \sqrt{\frac{9}{64\pi^2}}[2\cos^3\theta - 3\cos\theta\sin^2\theta] = \frac{3}{2}\sqrt{\frac{1}{7\pi}}(Y_{3,0}(\theta,\phi))$$

$$A_{3,\pm1}(\theta,\phi) = \mp\frac{\sqrt{3}}{2\pi}\left[\cos^2\theta + \frac{1}{4}(2\cos^2\theta - \sin^2\theta) + \frac{1}{8}\sin^2\theta\right]\sin\theta e^{\pm i\phi} = \mp\frac{3\sqrt{3}}{16\pi}[4\cos^2\theta - \sin^2\theta]\sin\theta e^{\pm i\phi} = \frac{3}{2}\sqrt{\frac{1}{7\pi}}Y_{3,\pm1}(\theta,\phi)$$

$$A_{3,\pm1}(\theta,\phi) = \mp\frac{3\sqrt{3}}{16\pi}[2\cos^2\theta + (3\cos^2\theta - 1)](\sin\theta e^{\pm i\phi}) = \mp\frac{3\sqrt{3}}{16\pi}[4\cos^2\theta - \sin^2\theta](\sin\theta e^{\pm i\phi}) = \frac{3}{2}\sqrt{\frac{1}{7\pi}}Y_{3,\pm1}(\theta,\phi)$$

$$A_{3,\pm2}(\theta,\phi) = \sqrt{\frac{15}{128\pi^2}}[2e^{\pm 2i\phi} + e^{\pm 2i\phi}]\cos\theta\sin^2\theta = \sqrt{\frac{135}{128\pi^2}}\cos\theta\sin^2\theta e^{\pm 2i\phi} = \frac{3}{2}\sqrt{\frac{1}{7\pi}}Y_{3,\pm2}(\theta,\phi)$$

$$A_{3,\pm3}(\theta,\phi) = \mp\sqrt{\frac{45}{256\pi^2}}(\sin^3\theta e^{\pm 3i\phi}) = \frac{3}{2}\sqrt{\frac{1}{7\pi}}Y_{3,\pm3}(\theta,\phi)$$

Again we produce a tensor with components being the normalized spherical harmonics scaled by the factor $\sqrt{9/(28\pi)}$,

$$A_{3,m}(\theta,\phi) = \frac{3}{2}\sqrt{\frac{1}{7\pi}}Y_{3,m}(\theta,\phi) \ .$$

Tensor Scalar Product

The scalar product of two irreducible tensors of rank ***l*** is defined as

$$\boldsymbol{A}_l \bullet \boldsymbol{T}_l = \sum_m^{\pm l}(-1)^m A_{lm}T_{l-m} = \sum_m^{\pm l}(-1)^m A_{l-m}T_{lm}. \qquad (9\text{-}21)$$

9.8.6 Hamiltonians from Tensors

Hamiltonians can usually be expressed in terms of scalar products of tensors. The main advantage in doing so is to utilize the well defined rotational properties of tensors, thus allowing for rotations of entire Hamiltonians. To demonstrate the tensor formalism we consider the interactions of nuclear spins, either with other nuclear spins or with an applied magnetic field. In the first case, the Hamiltonian between two spins can be written

$$\boldsymbol{H}(i,j) = \boldsymbol{I}_i \bullet \boldsymbol{A}_{ij} \bullet \boldsymbol{I}_j \,, \tag{9-22}$$

and in the latter, a spin interacting with an applied field $\boldsymbol{B}$, one may write

$$\boldsymbol{H}(i) = \boldsymbol{I}_i \bullet \boldsymbol{A}_i \bullet \boldsymbol{B} \,. \tag{9-23}$$

In both examples $\boldsymbol{I_i}$ is the spin angular momentum operator of spin $\boldsymbol{i}$ (a vector), and $\boldsymbol{A}$ is a matrix containing information about the interaction under consideration - strength and geometry. In matrix form these equations looks like

$$\boldsymbol{H}(i,j) = \begin{bmatrix} \boldsymbol{I}_{ix} & \boldsymbol{I}_{iy} & \boldsymbol{I}_{iz} \end{bmatrix} \bullet \begin{bmatrix} A_{xx} & A_{xy} & A_{xz} \\ A_{yx} & A_{yy} & A_{yz} \\ A_{zx} & A_{zy} & A_{zz} \end{bmatrix}_{ij} \bullet \begin{bmatrix} \boldsymbol{I}_{jx} \\ \boldsymbol{I}_{jy} \\ \boldsymbol{I}_{jz} \end{bmatrix} \tag{9-24}$$

and

$$\boldsymbol{H}(i) = \begin{bmatrix} \boldsymbol{I}_{ix} & \boldsymbol{I}_{iy} & \boldsymbol{I}_{iz} \end{bmatrix} \bullet \begin{bmatrix} A_{xx} & A_{xy} & A_{xz} \\ A_{yx} & A_{yy} & A_{yz} \\ A_{zx} & A_{zy} & A_{zz} \end{bmatrix}_{i} \bullet \begin{bmatrix} B_x \\ B_y \\ B_z \end{bmatrix} \tag{9-25}$$

which are compactly written as

$$\begin{aligned} \boldsymbol{H}(i) &= \sum_{u}^{axes} \sum_{v}^{axes} \langle 1|\boldsymbol{I}_i|u\rangle\langle u|\boldsymbol{A}_{ij}|v\rangle\langle v|\boldsymbol{I}_j|1\rangle \\ \boldsymbol{H}(i) &= \sum_{u}^{axes} \sum_{v}^{axes} \langle 1|\boldsymbol{I}_i|u\rangle\langle u|\boldsymbol{A}_{i}|v\rangle\langle v|\boldsymbol{B}|1\rangle \end{aligned} \tag{9-26}$$

with $u, v \in \{x, y, z\}$ and $\boldsymbol{B}$ a magnetic field vector. These equations can be rearranged to produce an equation involving two rank 2 Cartesian tensors by taking the dyadic product of the two vectors

involved.

$$\boldsymbol{H}(i,j) = \sum_{u}^{axes}\sum_{v}^{axes} \langle u|\boldsymbol{A}_{ij}|v\rangle\langle v|\boldsymbol{I}_j|1\rangle\langle 1|\boldsymbol{I}_i|u\rangle = \sum_{u}^{axes}\sum_{v}^{axes} \langle u|\boldsymbol{A}_{ij}|v\rangle\langle v|\boldsymbol{I}_j\boldsymbol{I}_i|u\rangle$$
$$\boldsymbol{H}(i) = \sum_{u}^{axes}\sum_{v}^{axes} \langle u|\boldsymbol{A}_{i}|v\rangle\langle v|\boldsymbol{B}|1\rangle\langle 1|\boldsymbol{I}_i|u\rangle = \sum_{u}^{axes}\sum_{v}^{axes} \langle u|\boldsymbol{A}_{i}|v\rangle\langle v|\boldsymbol{B}\boldsymbol{I}_i|u\rangle \tag{9-27}$$

The dyadic product to produce $\boldsymbol{BI_i}$ is explicitly done *via*

$$\begin{bmatrix} B_x \\ B_y \\ B_z \end{bmatrix} \bullet \begin{bmatrix} \boldsymbol{I}_{ix} & \boldsymbol{I}_{iy} & \boldsymbol{I}_{iz} \end{bmatrix} = \begin{bmatrix} B_x\boldsymbol{I}_{xi} & B_x\boldsymbol{I}_{yi} & B_x\boldsymbol{I}_{zi} \\ B_y\boldsymbol{I}_{xi} & B_y\boldsymbol{I}_{yi} & B_y\boldsymbol{I}_{zi} \\ B_z\boldsymbol{I}_{xi} & B_z\boldsymbol{I}_{yi} & B_z\boldsymbol{I}_{zi} \end{bmatrix}, \tag{9-28}$$

placing our two example treatments in the forms

$$\boldsymbol{H}(i,j) = \begin{bmatrix} A_{xx} & A_{xy} & A_{xz} \\ A_{yx} & A_{yy} & A_{yz} \\ A_{zx} & A_{zy} & A_{zz} \end{bmatrix}_{ij} \bullet \begin{bmatrix} \boldsymbol{I}_{xj}\boldsymbol{I}_{xi} & \boldsymbol{I}_{xj}\boldsymbol{I}_{yi} & \boldsymbol{I}_{xj}\boldsymbol{I}_{zi} \\ \boldsymbol{I}_{yj}\boldsymbol{I}_{xi} & \boldsymbol{I}_{yj}\boldsymbol{I}_{yi} & \boldsymbol{I}_{yj}\boldsymbol{I}_{zi} \\ \boldsymbol{I}_{zj}\boldsymbol{I}_{xi} & \boldsymbol{I}_{zj}\boldsymbol{I}_{yi} & \boldsymbol{I}_{zj}\boldsymbol{I}_{zi} \end{bmatrix} \tag{9-29}$$

and

$$\boldsymbol{H}(i) = \begin{bmatrix} A_{xx} & A_{xy} & A_{xz} \\ A_{yx} & A_{yy} & A_{yz} \\ A_{zx} & A_{zy} & A_{zz} \end{bmatrix}_{i} \bullet \begin{bmatrix} B_x\boldsymbol{I}_{xi} & B_x\boldsymbol{I}_{yi} & B_x\boldsymbol{I}_{zi} \\ B_y\boldsymbol{I}_{xi} & B_y\boldsymbol{I}_{yi} & B_y\boldsymbol{I}_{zi} \\ B_z\boldsymbol{I}_{xi} & B_z\boldsymbol{I}_{yi} & B_z\boldsymbol{I}_{zi} \end{bmatrix}. \tag{9-30}$$

An alternative treatment used in the latter case (single spin) is to rearrange (9-26) to produce an equation involving two rank 1 Cartesian tensors as opposed to rank 2 tensors.

$$\boldsymbol{H}(i) = \sum_{u}^{axes}\sum_{v}^{axes} \langle 1|\boldsymbol{I}_i|u\rangle\langle u|\boldsymbol{A}_i|v\rangle\langle v|\boldsymbol{B}|1\rangle = \sum_{u}^{axes} \langle 1|\boldsymbol{I}_i|u\rangle\langle u|\boldsymbol{A}_i\boldsymbol{B}|1\rangle \tag{9-31}$$

The matrix form in the case is the product of two vectors.

$$\boldsymbol{H}(i) = \begin{bmatrix} \boldsymbol{I}_{ix} & \boldsymbol{I}_{iy} & \boldsymbol{I}_{iz} \end{bmatrix} \bullet \begin{bmatrix} A_{xx}B_x + A_{xy}B_y + A_{xz}B_z \\ A_{yx}B_x + A_{yy}B_y + A_{yz}B_z \\ A_{zx}B_x + A_{zy}B_y + A_{zz}B_z \end{bmatrix}. \tag{9-32}$$

This equation along with (9-29) can be written generally in terms of two Cartesian tensors, one in-

volving only spatial coordinates $\boldsymbol{A}$ and one involving only spin coordinates $\boldsymbol{T}$. From (9-29)

$$\boldsymbol{H}_{ij} = \boldsymbol{A}_{ij} \bullet \boldsymbol{T}_{ij} \tag{9-33}$$

and from (9-32)

$$\boldsymbol{H}_{i} = \boldsymbol{A}_{i} \bullet \boldsymbol{T}_{i}. \tag{9-34}$$

The separation of the Hamiltonian into spatial and spatial tensor components allows one to easily rotate $\boldsymbol{H}$ in either coordinate or spin space. Similar notation is also utilized to describe equation (9-30), but in this instance there are spatial components in both tensors. One must then be careful when performing rotations in this particular case.

10 Spin Tensors

10.1 Overview

Class SPIN TENSOR (spin_T) contains routines to handle general tensors involving spin angular momentum. Each spin tensor is maintained in its irreducible spherical form, each component being a spin operator. Functions are provided for creating spin tensors, for taking the cross product between tensors, and for taking the scalar product of tensors. Standard output is supplied as well.

10.2 Available Functions

Basic Functions

Spin Tensor Functions with Scalars

Rank 1 Spin Tensor Functions

Rank 2 Spin Tensor Functions

Spin Tensor Complex Functions

Spin Tensor Auxiliary Functions

Spin tensor Parameter Set and I/O Functions

10.3 Spin Tensor Discussion

10.4 Spin Tensor Figures

10.5 Spin Tensor Programs

10.6 Basic Routines

10.6.1 spin_T

Usage:

#include <spin_T.h>
spin_T ()
spin_T (spin_sys &sys)
spin_T (spin_T &SphT)
spin_T (spin_T &SphT, int l)

Description:

The function *spin_T* is used to create a spin tensor quantity. There are currently four methods for creating a new spin tensor.

1. spin_T () - When *spin_T* is invoked without arguments it creates an empty (NULL) spin tensor.
2. spin_T (spin_sys &sys) - When *spin_T* is invoked with a spin system in the argument it creates an empty spin tensor associated with the spin system given. The resultant spin tensor can be subsequently used with all spin tensor functions.
3. spin_T(spin_op &SOp) - When *spin_T* is used with another spin tensor as its argument a new spin tensor is produced that is identical to the spin tensor given.
4. spin_T(spin_op &SOp, int l) - When *spin_T* is used with another spin tensor and a rank index l as its arguments a new spin tensor is produced that contains only the irreducible rank l tensor of the spin tensor given.

Return Value:

The function (constructor) *spin_T* returns no values. It is used strictly to create a new spin tensor.

Example(s):

```
#include <gamma.h>
main()
  {
  spin_T SphT();                    // create a NULL spin tensor named sphT.
  spin_sys ABX(3);                  // create a three spin system called ABX.
  spin_T SphT1(ABX);                // create spin tensor SphT1 associated with ABX.
  spin_T SphT2(SphT);               // create a new spin tensor SphT2 from SphT1.
  SphT = T2(ABX, 0, 2);             // assign SphT to be a rank 2 spin tensor, spins 0&2
  SphT1 = T_mult(SphT, Sph_T);      // reset SphT1 to rank 4 tensor, prod. of SphT twice.
  SphT3(SphT,2);                    // tensor SphT3 from SphT1 rank 2 irreducible part.
  }
```

See Also: =

10.6.2 =

Usage:

#include <spin_T.h>
spin_T operator = (spin_T &SphT);

Description:

The assignment operator (=) provides the use of equality in the algebraic manipulations of spin tensors. The user may set one spin tensor equal to another spin tensor.

Return Value:

None, this function is void. The mathematical operation is performed.

Example(s):

```
#include <gamma.h>
main()
  {
  spin_sys AX(2);                  // create a two spin system called AX.
  spin_T SphT(AX);                 // new spin tensor associated with spin system AX.
  SphT = T2(AX, 0, 1);             // set SphT to the rank 2 spin tensor for AX system.
  spin_T SphT1();                  // create another spin tensor, NULL.
  SphT1 = SphT;                    // set SphT1 equal to SphT.
  }
```

See Also: spin_T

10.7 Rank 1 Spin Tensors

10.7.1 T1

Usage:

```
#include <spin_T.h>
spin_T T1(spin_sys& sys, int spin);
spin_op T1(spin_sys& sys, int spin, int l, int m);
```

Description:

The function *T1* returns either the standard rank 1 spin tensor or a specific component of that tensor.

1. T1 (spin_sys& sys, int spin) - When ***T1*** is invoked with a spin system and a spin index as arguments it creates the standard spherical rank 1 spin tensor.
2. T1(spin_sys &sys, int spin, int l, int m) - When ***T1*** is invoked with a spin system, a spin index, and angular momentum components the function returns a spin operator which is the l,m irreducible spherical component of the standard rank 1 spin tensor.

The rank 1 spin tensor consists of four irreducible spherical components specified by

$$T^{(1)}_{l,\,m}(i) \qquad l \in [0,\,1] \qquad m \in [-l,\,l]\ ,$$

where i is the spin index. The tensor index *l* spans the rank, in this case 1, while the tensor index *m* spans *l*. All tensor components are returned as spin operators, these are mathematically defined by equation (10-12).

$$T^{(1)}_{0,\,0}(i) = 0 \qquad \boldsymbol{T}_{1,\,-1} = \frac{1}{\sqrt{2}}\boldsymbol{J}_{+} \qquad \boldsymbol{T}_{1,\,0} = \boldsymbol{J}_{z} \qquad \boldsymbol{T}_{1,\,1} = -\frac{1}{\sqrt{2}}\boldsymbol{J}_{-}$$

Return Value:

A rank 1 irreducible spherical spin tensor.

Example(s):

```
#include <gamma.h>
main()
  {
  spin_sys AMX(3);                  // create a three spin system called AMX.
  spin_op SOp;                      // create spin operator associated with AMX, SOp.
  spin_T SphT(AMX);                 // create a spin tensor associated with AMX, SphT.
  SphT = T1(AMX, 0);                // SphT is the rank 1 spin tensor for spin 0.
  SOp = T1(AMX, 0, 1, 0);           // SOp is the 1,0 rank 1 spin tensor component spin 0.
  cout << SOp;                      // prints SOp, 1,1 component of the spin tensor.
  }
```

See Also: T2

10.8 Rank 2 Spin Tensors

10.8.1 T2

Usage:

```
#include <spin_T.h>
spin_T T2(spin_sys &sys, int spin1, int spin2);
spin_T T2(spin_sys &sys, spin_op &Im1, spin_op &Iz1, spin_op &Ip1,
                                        spin_op &Im2, spin_op &Iz2, spin_op &Ip2);
spin_op T2(spin_sys &sys, int spin1, int spin2, int l, int m);
spin_op T2(spin_op &Im1, spin_op &Iz1, spin_op &Ip1,
                                    spin_op &Im2, spin_op &Iz2, spin_op &Ip2, int l, int m);
```

Description:

The function ***T2*** returns either the standard rank 2 spin tensor or a specific component of that tensor. This tensor is equivalent to that from the product of two rank 1 spin tensors.

1. T2 (spin_sys &sys, int spin1, int spin2) - When ***T2*** is invoked with only a spin system, ***sys***, and spin indices, ***spin1*** and ***spin 2***, in the argument list it creates the standard irreducible spherical rank 2 spin tensor containing the nine components of equation sosi.
2. T2(spin_sys &sys, spin_op &Im1, spin_op &Iz1, spin_op &Ip1, spin_op &Im2, spin_op &Iz2, spin_op &Ip2) - This function performs the equivalent to 1. above except that the spin operators involved (these are components of the two rank 1 spin tensors) are directly input. This avoids any repetitive computation of the spin operators on successive calls to the function. ***Im1*** is spin operator I-, ***Iz1*** is the spin operator Iz and ***Ip1*** the spin operator I_+, all for the first spin. ***Im2***, ***Iz2***, and ***Ip2*** are the spin operators for the second spin.
3. T2(spin_sys &sys, int spin1, int spin2, int l, int m) - When *T2* is invoked with a spin system, spin indices, and angular momentum components l & m, the function returns a spin operator which is the l,m irreducible spherical component of the standard rank 2 spin tensors. The appropriate component of equation sosi is the result.
4. T2(spin_sys &sys, spin_op &Im1, spin_op &Iz1, spin_op &Ip1, spin_op &Im2, spin_op &Iz2, spin_op &Ip2, int l, int m) - This function performs the equivalent to 3. above except that the spin operators involved are directly input. This avoids any repetitive computation of the spin operators on successive calls to the function. Im1 is spin operator I_-, Iz1 is the spin operator I_z and Ip1 the spin operator I_+, all for the first spin. Im2, Iz2, and Ip2 are the spin operators for the second spin.

The rank 2 spin tensor consists of nine irreducible spherical components specified as either

$$T_{l,m}(i,j) \quad \text{or} \quad T_{l,m}(i),$$

where ***i*** and ***j*** are spin indices. The tensor index ***l*** spans the rank, in this case 2, while the tensor index ***m*** spans ***l***,

$$l \in [0, 2] \quad \& \quad m \in [-l, l],$$

All tensor components are returned as spin operators.

Return Value:

A rank 2 irreducible spherical spin tensor.

Example(s):

```
#include <gamma.h>
main()
  {
  spin_sys AMX(3);                  // create a three spin system called AMX.
  spin_op SOp(AMX);                 // create a spin operator associated with AMX, SOp.
  spin_T SphT(AMX);                 // create a spin tensor associated with AMX, SphT.
  SphT = T1(AMX, 0);                // SphT is the rank 1 spin tensor for spin 0.
  SOp = T1(AMX, 0, 1, 0);           // SOp is the 1,0 rank 1 spin tensor component spin 0.
  cout << SOp;                      // prints SOp, the 1,1 component of spin tensor.
  }
```

Mathematical Basis:

A general rank 2 tensor contains nine components. When this tensor involves two spin components of angular momenta its irreducible spherical components are specified as $T^{(2)}_{l,m}(i,j)$, the formulas

which produce them given by equation on page 192[1],

$$T_{0,0}(ij) = \frac{-1}{\sqrt{3}}\left[I_{iz}I_{jz} + \frac{1}{2}(I_{i+}I_{j-} + I_{i-}I_{j+})\right]$$

$$T_{1,0}(ij) = \frac{-1}{2\sqrt{2}}[I_{i+}I_{j-} + I_{i-}I_{j+}] \qquad T_{1,\pm1}(ij) = \frac{-1}{2}[I_{i\pm}\ I_{jz} + I_{iz}I_{j\pm}\ ]$$

$$T_{2,0}(ij) = \frac{1}{\sqrt{6}}[3I_{iz}I_{jz} - (\boldsymbol{I}_i \bullet \boldsymbol{I}_j)]$$

$$T_{2,\pm1}(ij) = \mp\frac{1}{2}[I_{i\pm}\ I_{jz} + I_{iz}I_{j\pm}\ ] \qquad T_{2,\pm2}(ij) = \frac{1}{2}[I_{i\pm}\ I_{j\pm}\ ]$$

When treating spin angular momentum where both spins have I=1/2 the matrix representations of these tensor components (operators) are given by the following. They are shown in the composite

1. Readers should note that there are several different conventions found in the literature used in expressing these tensor components. Therefore these formulas may vary from those given by other sources. The following reasons will likely explain any discrepancies found between the formulae here and other references:
1.) All components differ by a constant - this is mathematically valid because a constant multiple does not change the tensor properties, it is just a normalization factor difference. Equation has a normalization in which these components result from a product of two of rank 1 tensors given in equation (10-12) as shown in the discussion Section at the end of this Chapter.
2.) The spin operators are different - most differences result either from use of I_x & I_y *versus* I_+ & I_- or from the explicit expansion of the product of $\mathbf{I}_i \bullet \mathbf{I}_j$. For example, the $T_{0,0}$ component formula given shows three common variations due to these differences. Another explanations is that some authors (like Haeberlen) use specially defined operators I_+ & I_- which make their formulations look different. A third more direct reason for a formula discrepancy is that the components being compared are not the same, i.e. one may accidentally be comparing $T_{1,0}$ above with the $T_{1,0}$ obtained directly from a rank 1 treatment or the single spin variant of the rank 2 treatment. While similar in appearance, they are scaled differently: compare equation (10-12) and equation .)
3.) The components differ by several constants - if one is always paring up these tensor components with spatial tensor components it is allowable to scale one as long as compensation is made in the other (See EBW, the bottom of page 47 for an example). This is not recommended. One reason is that the tensor (either space or spin) will no longer rotate properly. Another reason is that the components no longer relate to one another via the ladder operators. Equation (10-11), which defines the irreducible spherical tensor, is invalid in such a treatment.

Hilbert space of the two spins (spin indices are implicit here).

Rank 2 Irreducible Spherical Spin Tensor Components
Matrix Representations in 2-spin (I=1/2) Hilbert Space

$$T^{(2)}_{0,0} = \frac{-1}{4\sqrt{3}}\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 2 & 0 \\ 0 & 2 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad T^{(2)}_{1,0} = \frac{-1}{2\sqrt{2}}\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \quad T^{(2)}_{1,-1} = \frac{1}{4}\begin{bmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & 1 & -1 & 0 \end{bmatrix} \quad T^{(2)}_{1,1} = \frac{1}{4}\begin{bmatrix} 0 & 1 & -1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & -1 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$$T^{(2)}_{2,0} = \frac{1}{2\sqrt{6}}\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & -1 & 0 \\ 0 & -1 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad T^{(2)}_{2,1} = \frac{1}{4}\begin{bmatrix} 0 & -1 & -1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} \quad T^{(2)}_{2,-1} = \frac{1}{4}\begin{bmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & -1 & -1 & 0 \end{bmatrix} \quad T^{(2)}_{2,-2} = \frac{1}{2}\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \quad T^{(2)}_{2,2} = \frac{1}{2}\begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

See Also: T1, T2SS

10.8.2 T20

Usage:

```
#include <spin_T.h>
spin_T T20(spin_sys &sys, int spin1, int spin2, int m);
spin_T T20(spin_op &Im1, spin_op &Iz1, spin_op &Ip1, spin_op &Im2,
                                    spin_op &Iz2, spin_op &Ip2, int m);
```

Description:

The function ***T20*** returns either the standard irreducible rank 2 spin tensor or a specific component of that tensor. For this function the only allowed value of *m* is 0.

1. T20(spin_sys &sys, int spin1, int spin2, int m) - When *T20* is invoked with a spin system, ***sys***, two spin indices, ***spin1 and spin2***, and a angular momentum component, ***m***, the function returns an irreducible spin-space spherical rank 2 tensor containing the single component of equation on page 192.
2. T20(spin_sys &sys, spin_op &Im, spin_op &Iz, spin_op &Ip, coord &pt, int rev=0) - This function performs the equivalent to 1. above except that the spin operators involved are directly input. This avoids any repetitive computation of the spin operators on successive calls to the function. ***Im*** is spin operator I_-, ***Iz*** is the spin operator I_z and ***Ip*** the spin operator I_+, all for the spin being treated.

The irreducible rank 0 part of the general rank 2 spin tensor consists of only one spherical components specified as

$$T^{(2)}_{0,m}(i,j)\ ,$$

where i is the spin index. The tensor index the tensor index *m* spans the rank.

$$m \in [-0, 0]$$

All tensor components are returned as spin operators.

Return Value:

An irreducible rank 2 space/spin tensor or a component of that tensor.

Example(s):

```
#include <gamma.h>
main()
  {
  SOp = T1(AMX, 0, 1, 0);          // SOp is the 1,0 rank 1 spin tensor component spin 0.
  cout << SOp;                     // prints SOp, the 1,1 component of spin tensor.
  }
```

Mathematical Basis:

This tensor component is given by the following equation,

Rank 2 Spin Tensor Irreducible Spherical 0,0 Components

$$T^{(2)}_{0,0}(i,j) = \frac{-1}{\sqrt{3}}\hat{\boldsymbol{I}}_i \bullet \hat{\boldsymbol{I}}_j$$

See Also: T1, T2

10.8.3 T21

Usage:

```
#include <spin_T.h>
spin_T T21(spin_sys &sys, int spin1, int spin2, int m);
spin_T T21(spin_op &Im1, spin_op &Iz1, spin_op &Ip1, spin_op &Im2,
                                    spin_op &Iz2, spin_op &Ip2, int m);
```

Description:

The function ***T21*** returns one of the ***l=1*** components of the rank 2 spin tensor formed from the product of a rank 1 spatial tensor with a rank 1 spin tensor.

1. T21(spin_sys &sys, int spin, coord &pt, int rev=0) - When ***T21*** is invoked with a spin system, ***sys***, a spin index, ***spin*** and a coordinate point, ***pt***, the function returns an irreducible spin-space spherical rank 2 tensor containing the five components of equation on page 192.
2. T21(spin_sys &sys, spin_op &Im, spin_op &Iz, spin_op &Ip, coord &pt, int rev=0) - This function performs the equivalent to 1. above except that the spin operators involved are directly input. This avoids any repetitive computation of the spin operators on successive calls to the function. ***Im*** is spin operator I_-, ***Iz*** is the spin operator I_z and ***Ip*** the spin operator I_+, all for the spin being treated.

The general rank 2 space/spin tensor contains of three rank 1 irreducible spherical components specified as

$$T^{(2)}_{1,m}(i,j),$$

where ***i*** is the spin index. The tensor index the tensor index *m* spans the rank, $\boldsymbol{m \in [-1, 1]}$.

All tensor components are returned as spin operators.

Return Value:

An spin operator.

Example:

```
#include <gamma.h>
main()
  {
  spin_sys AMX(3);                    // create a three spin system called AMX.
  cout << SOp;                        // prints SOp, the 1,1 component of spin tensor.
  }
```

Mathematical Basis:

The three tensor components returned by this function are specified by the following equations.

Rank 2 Spin Tensor Irreducible Spherical 1,m Components

$$T^{(2)}_{1,0}(i,j) = \frac{-1}{2\sqrt{2}}[\boldsymbol{I}_{i+}\boldsymbol{I}_{j-} - \boldsymbol{I}_{i-}\boldsymbol{I}_{j+}] \qquad T^{(2)}_{1,\pm1}(i,j) = \frac{-1}{2}[\boldsymbol{I}_{i\pm}\ \boldsymbol{I}_{jz} - \boldsymbol{I}_{iz}\boldsymbol{I}_{j\pm}\ ]$$

See Also: T1, T2

10.8.4 T22

Usage:

```
#include <spin_T.h>
spin_T T22(spin_sys &sys, int spin1, int spin2, int m);
spin_T T22(spin_op &Im1, spin_op &Iz1, spin_op &Ip1, spin_op &Im2,
                                    spin_op &Iz2, spin_op &Ip2, int m);
```

Description:

The function ***T22*** returns either the standard irreducible rank 2 space/spin tensor or a specific component of that tensor.

1. T21SS(spin_sys &sys, int spin, coord &pt, int rev=0) - When *T2* is invoked with a spin system, ***sys***, a spin index, ***spin*** and a coordinate point, ***pt***, the function returns an irreducible spin-space spherical rank 2 tensor containing the five components of equation on page 192.
2. T21SS(spin_sys &sys, spin_op &Im, spin_op &Iz, spin_op &Ip, coord &pt, int rev=0) - This function performs the equivalent to 1. above except that the spin operators involved are directly input. This avoids any repetitive computation of the spin operators on successive calls to the function. ***Im*** is spin operator I_-, ***Iz*** is the spin operator I_z and ***Ip*** the spin operator I_+, all for the spin being treated.

The irreducible rank 2 space/spin tensor consists of five spherical components specified as

$$T^{(2)}_{2,m}(i,j)\ ,$$

where i is the spin index. The tensor index the tensor index *m* spans the rank, $\boldsymbol{m \in [-2, 2]}$.

All tensor components are returned as spin operators.

Return Value:

An irreducible rank 2 space/spin tensor or a component of that tensor.

Example(s):

```
#include <gamma.h>
main()
  {
  spin_sys AMX(3);                    // create a three spin system called AMX.
  cout << SOp;                        // prints SOp, the 1,1 component of spin tensor.
  }
```

Mathematical Basis:

The five tensor components are given by the following equations,.

Rank 2 Spin Tensor Irreducible Spherical 2,m Components

$$T^{(2)}_{2,0}(i) = \frac{1}{\sqrt{6}}[3\boldsymbol{I}_{iz}\boldsymbol{I}_{jz} - (\vec{\boldsymbol{I}}_i \bullet \vec{\boldsymbol{I}}_j)]$$

$$T^{(2)}_{2,\pm1}(i) = \mp\frac{1}{2}[\boldsymbol{I}_{i\pm}\ \boldsymbol{I}_{jz} + \boldsymbol{I}_{iz}\boldsymbol{I}_{j\pm}\ ] \qquad T^{(2)}_{2,\pm2}(i) = \frac{1}{2}[\boldsymbol{I}_{i\pm}\ \boldsymbol{I}_{j\pm}\ ]$$

See Also: T1, T2

10.8.5 T2SS

Usage:

```
#include <spin_T.h>
spin_T T2SS(spin_sys &sys, int spin, coord &pt, int rev=0);
spin_T T2SS(spin_sys &sys, spin_op &Im, spin_op &Iz, spin_op &Ip, coord &pt, int rev=0);
spin_op T2SS(spin_sys &sys, int spin, coord &pt, int l, int m, int rev=0);
spin_op T2SS(spin_sys &sys, spin_op &Im, spin_op &Iz, spin_op &Ip,
                                                    coord &pt, int l, int m, int rev=0);
```

Description:

The function ***T2SS*** returns either the standard rank 2 space/spin tensor or a specific component of that tensor.

1. T2SS(spin_sys &sys, int spin, coord &pt, int rev=0) - When *T2* is invoked with a spin system, ***sys***, a spin index, ***spin*** and a coordinate point, ***pt***, the function returns a spin-space irreducible spherical rank 2 tensor containing the nine components of equation on page 192. This rank 2 tensor corresponds to a single spin because it is formulated from the product spin angular momentum vector (rank1 spin tensor) and a spatial vector (rank 1 spatial tensor) as given by the coordinate point. The coordinate is assumed to represent a normalized Cartesian vector. If the flag ***rev*** is set to non-zero, the tensor is assumed to be the product of the spatial vector times the spin vector as opposed to the spin vector times the spatial vector (default).
2. T2SS(spin_sys &sys, spin_op &Im, spin_op &Iz, spin_op &Ip, coord &pt, int rev=0) - This function per-

forms the equivalent to 1. above except that the spin operators involved are directly input. This avoids any repetitive computation of the spin operators on successive calls to the function. ***Im*** is spin operator I_-, ***Iz*** is the spin operator I_z and ***Ip*** the spin operator I_+, all for the spin being treated.

3. T2SS(spin_sys &sys, int spin, coord &pt, int l, int m, int rev=0) - This function returns the ***l***,***m*** irreducible spherical component of the tensor formed from 1. above. Arguments ***l*** & ***m*** are the tensor angular momentum components. A spin operator which is the appropriate component of equation is returned.
4. T2SS(spin_sys &sys, spin_op &Im, spin_op &Iz, spin_op &Ip, coord &pt, int l, int m, int rev=0) - This function returns the ***l***,***m*** irreducible spherical component of the tensor formed from 2. above. Arguments ***l*** & ***m*** are the tensor angular momentum components. A spin operator which is the appropriate component of equation is returned.

The rank 2 space/spin tensor consists of nine irreducible spherical components specified as

$$T^{(2)}_{l,\,m}(i)\ ,$$

where ***i*** is the spin index. The tensor index ***l*** spans the rank, in this case 2, while the tensor index ***m*** spans ***l***, $l \in [0, 2]$ and $m \in [-l, l]$. All tensor components are returned as spin operators.

Return Value:

A rank 2 irreducible spherical spin tensor.

Example(s):

```
#include <gamma.h>
main()
  {
  spin_sys AMX(3);                  // create a three spin system called AMX.
  spin_op SOp(AMX);                 // create a spin operator associated with AMX, SOp.
  spin_T SphT(AMX);                 // create a spin tensor associated with AMX, SphT.
  SphT = T1(AMX, 0);                // SphT is the rank 1 spin tensor for spin 0.
  SOp = T1(AMX, 0, 1, 0);           // SOp is the 1,0 rank 1 spin tensor component spin 0.
  cout << SOp;                      // prints SOp, the 1,1 component of spin tensor.
  }
```

Mathematical Basis:

There are occasions when it is convenient to utilize a rank 2 “spin” tensor which actually contains both spatial and spin terms. In this case the tensor will involve only one spin, its nine components are specified as

$$T^{(2)}_{l,\,m}(i)\ .$$

This tensor if formed from the product of a rank 1 spin tensor with a rank 1 spatial tensor. The nine

formulas which produce these components are given by[1] equations

Rank 2 Irreducible Spherical Spin-Space Tensor Components

$$T^{(2)}_{0,0}(i) = \frac{-1}{\sqrt{3}}\left[\boldsymbol{I}_{iz}u_z + \frac{1}{2}(\boldsymbol{I}_{i+}u_- + \boldsymbol{I}_{i-}u_+)\right] = \frac{-1}{\sqrt{3}}\grave{\boldsymbol{I}}_i \bullet \grave{\boldsymbol{u}}$$

$$T^{(2)}_{1,0}(i) = \frac{-1}{2\sqrt{2}}[\boldsymbol{I}_{i+}u_- - \boldsymbol{I}_{i-}u_+] \qquad T^{(2)}_{1,\pm1}(i) = \frac{-1}{2}[\boldsymbol{I}_{i\pm}\,u_z - \boldsymbol{I}_{iz}u_\pm\,]$$

$$T^{(2)}_{2,0}(i) = \frac{1}{\sqrt{6}}[3\boldsymbol{I}_{iz}u_z - (\grave{\boldsymbol{I}}_i \bullet \grave{\boldsymbol{u}})]$$

$$T^{(2)}_{2,\pm1}(i) = \mp\frac{1}{2}[\boldsymbol{I}_{i\pm}\,u_z + \boldsymbol{I}_{iz}u_\pm\,] \qquad T^{(2)}_{2,\pm2}(i) = \frac{1}{2}[\boldsymbol{I}_{i\pm}\,u_\pm\,]$$

where $\grave{\boldsymbol{u}}$ is a normalized vector $\grave{\boldsymbol{u}} = u_x\boldsymbol{i} + u_y\boldsymbol{j} + u_z\boldsymbol{k}$. Of course, setting the vector $\grave{\boldsymbol{u}}$ equal to the angular momentum (vector) operator $\grave{\boldsymbol{I}}_j$ simply regenerates equations (18-3). The simplest situation mathematically occurs when $\grave{\boldsymbol{u}}$ points along the positive z-axis, $\grave{\boldsymbol{u}} = \grave{\boldsymbol{k}}$. Put another way, the tensor simplifies when placed in a coordinates system whose z-axis is aligned with the z-axis of the spatial tensor (vector). The applicable equations in this case are shown in the following figure.

Rank 2 Irreducible Spherical Spin-Space Tensor Components Aligned Along the z-axis of the Spatial Vector

$$T^{(2)}_{0,0}(i) = \frac{-1}{\sqrt{3}}\boldsymbol{I}_{iz}$$

$$T^{(2)}_{1,0}(i) = 0 \qquad T^{(2)}_{1,\pm1}(i) = \frac{-1}{2}\boldsymbol{I}_{i\pm}$$

$$T_{2,0}(ij) = \frac{2}{\sqrt{6}}\boldsymbol{I}_{iz}$$

$$T^{(2)}_{2,\pm1}(i) = \mp\frac{1}{2}\boldsymbol{I}_{i\pm} \qquad T^{(2)}_{2,\pm2}(i) = 0$$

For a spin 1/2 particle and $\grave{\boldsymbol{u}}$ along the positive z-axis the matrix form of these tensor components

1. Bear in mind that these formulae have the **2nd** tensor as the spatial one. A sign change will occur for the ***l=1*** components if $\grave{\boldsymbol{u}}$ is set to the 1st and the spin tensor to the second.

are shown in the following figure[1] (in the single spin Hilbert space).

Rank 2 Irreducible Spherical Spin-Space Tensor Components Matrix Representations in 1-spin (I=1/2) Hilbert Space

$$T^{(2)}_{0,0} = \frac{-1}{2\sqrt{3}}\begin{bmatrix}1 & 0\\ 0 & -1\end{bmatrix} \qquad T^{(2)}_{1,0} = \begin{bmatrix}0 & 0\\ 0 & 0\end{bmatrix} \qquad T^{(2)}_{1,-1} = \frac{-1}{2}\begin{bmatrix}0 & 0\\ 1 & 0\end{bmatrix} \qquad T^{(2)}_{1,1} = \frac{-1}{2}\begin{bmatrix}0 & 1\\ 0 & 0\end{bmatrix}$$

$$T^{(2)}_{2,0} = \frac{1}{\sqrt{6}}\begin{bmatrix}1 & 0\\ 0 & -1\end{bmatrix} \quad T^{(2)}_{2,1} = \frac{-1}{2}\begin{bmatrix}0 & 1\\ 0 & 0\end{bmatrix} \quad T^{(2)}_{2,-1} = \frac{1}{2}\begin{bmatrix}0 & 0\\ 1 & 0\end{bmatrix} \quad T^{(2)}_{2,-2} = \begin{bmatrix}0 & 0\\ 0 & 0\end{bmatrix} \quad T^{(2)}_{2,2} = \begin{bmatrix}0 & 0\\ 0 & 0\end{bmatrix}$$

One must be very careful in using single spin rank 2 tensors of this type because they contain both spatial and spin components. It is improper to rotate this tensor in spin space because it also rotates spatial variables. If some Hamiltonian is a product of a spatial tensor and this tensor, the spatial tensor cannot be rotated as it rotates only part of the spatial components[2].

See Also: T1, T2

10.8.6 T20SS

Usage:

```
#include <spin_T.h>
spin_T T20SS(spin_sys &sys, int spin, coord &pt, int m, int rev=0);
spin_T T20SS(spin_sys &sys, spin_op &Im, spin_op &Iz, spin_op &Ip, coord &pt, int m, int rev=0);
```

Description:

The function ***T20SS*** returns either the standard irreducible rank 2 space/spin tensor or a specific component of that tensor. For this function the only allowed value of *m* is 0. Furthermore the argument ***rev*** is not utilized. Both of these arguments are included for consistency with the functions ***T21SS*** and ***T22SS***.

1. T20SS(spin_sys &sys, int spin, coord &pt, int rev=0) - When *T2* is invoked with a spin system, ***sys***, a spin index, ***spin*** and a coordinate point, ***pt***, the function returns an irreducible spin-space spherical rank 2 tensor containing the five components of equation on page 192. This irreducible rank 2 tensor corresponds to a single spin because it is formulated from the product spin angular momentum vector (rank1 spin tensor) and a spatial vector (rank 1 spatial tensor) as given by the coordinate point. The coordinate is assumed to represent a normalized Cartesian vector. If the flag ***rev*** is set to non-zero, the tensor is assumed to be the product of the spatial vector times the spin vector as opposed to the spin vector times the

1. The GAMMA program which produced these matrix representations can be found at the end of this Chapter, Rank2SS_SpinT.cc.
2. See the discussion in Mehring.

spatial vector (default).

2. T20SS(spin_sys &sys, spin_op &Im, spin_op &Iz, spin_op &Ip, coord &pt, int rev=0) - This function performs the equivalent to 1. above except that the spin operators involved are directly input. This avoids any repetitive computation of the spin operators on successive calls to the function. ***Im*** is spin operator I_-, ***Iz*** is the spin operator I_z and ***Ip*** the spin operator I_+, all for the spin being treated.

The irreducible rank 2 space/spin tensor consists of five spherical components specified as

$$T^{(2)}_{0,m}(i)\ ,$$

where i is the spin index. The tensor index the tensor index ***m*** spans the rank, $\boldsymbol{m \in [-0, 0]}$. All tensor components are returned as spin operators.

Return Value:

An irreducible rank 2 space/spin tensor or a component of that tensor.

Example(s):

```
#include <gamma.h>
main()
  {
  spin_sys AMX(3);                  // create a three spin system called AMX.
  spin_op SOp(AMX);                 // create a spin operator associated with AMX, SOp.
  spin_T SphT(AMX);                 // create a spin tensor associated with AMX, SphT.
  SphT = T1(AMX, 0);                // SphT is the rank 1 spin tensor for spin 0.
  SOp = T1(AMX, 0, 1, 0);           // SOp is the 1,0 rank 1 spin tensor component spin 0.
  cout << SOp;                      // prints SOp, the 1,1 component of spin tensor.
  }
```

Mathematical Basis:

This tensor component is given by the following equation,

Rank 2 Spin-Space Tensor Irreducible Spherical 0,0 Component

$$\left\{ T^{(2)}_{0,0}(i) = \frac{-1}{\sqrt{3}} \grave{\boldsymbol{I}}_i \bullet \grave{\boldsymbol{u}} \right\}_{\substack{\text{z-axis} \\ \text{alignment}}} \rightarrow \frac{-1}{\sqrt{3}} \boldsymbol{I}_{iz}$$

where $\grave{\boldsymbol{u}}$ is a normalized vector $\grave{\boldsymbol{u}} = u_x\boldsymbol{i} + u_y\boldsymbol{j} + u_z\boldsymbol{k}$.

See Also: T1, T2

10.8.7 T21SS

Usage:

```
#include <spin_T.h>
spin_T T21SS(spin_sys &sys, int spin, coord &pt, int m, int rev=0);
spin_T T21SS(spin_sys &sys, spin_op &Im, spin_op &Iz, spin_op &Ip, coord &pt, int m, int rev=0);
```

Description:

The function ***T21SS*** returns one of the $l = 1$ components of the rank 2 space/spin tensor formed from the product of a rank 1 spatial tensor with a rank 1 spin tensor.

1. T21SS(spin_sys &sys, int spin, coord &pt, int rev=0) - When ***T21SS*** is invoked with a spin system, ***sys***, a spin index, ***spin*** and a coordinate point, ***pt***, the function returns an irreducible spin-space spherical rank 2 tensor containing the five components of equation on page 192. This irreducible rank 2 tensor corresponds to a single spin because it is formulated from the product spin angular momentum vector (rank1 spin tensor) and a spatial vector (rank 1 spatial tensor) as given by the coordinate point. The coordinate is assumed to represent a normalized Cartesian vector. If the flag ***rev*** is set to non-zero, the tensor is assumed to be the product of the spatial vector times the spin vector as opposed to the spin vector times the spatial vector (default).
2. T20SS(spin_sys &sys, spin_op &Im, spin_op &Iz, spin_op &Ip, coord &pt, int rev=0) - This function performs the equivalent to 1. above except that the spin operators involved are directly input. This avoids any repetitive computation of the spin operators on successive calls to the function. ***Im*** is spin operator I_-, ***Iz*** is the spin operator I_z and ***Ip*** the spin operator I_+, all for the spin being treated.

The general rank 2 space/spin tensor contains of three rank 1 irreducible spherical components specified as

$$T^{(2)}_{1,m}(i) ,$$

where i is the spin index. The tensor index the tensor index ***m*** spans the rank, ***m*** $\in$ ***[-1, 1]***. All tensor components are returned as spin operators.

Return Value:

An spin operator.

Example:

```
#include <gamma.h>
main()
  {
  spin_sys AMX(3);                  // create a three spin system called AMX.
  cout << SOp;                      // prints SOp, the 1,1 component of spin tensor.
  }
```

Mathematical Basis:

The three tensor components returned by this function are specified by the following equations,

Rank 2 Spin-Space Tensor Irreducible Spherical 1,m Components

$$\left\{ T^{(2)}_{1,0}(i) = \frac{-1}{2\sqrt{2}}[\boldsymbol{I}_{i+}u_{-} - \boldsymbol{I}_{i-}u_{+}] \right\}_{\substack{\text{z-axis} \\ \text{alignment}}} \rightarrow 0$$

$$\left\{ T^{(2)}_{1,\pm 1}(i) = \frac{-1}{2}[\boldsymbol{I}_{i\pm}\ u_z - \boldsymbol{I}_{iz}u_{\pm}\ ] \right\}_{\substack{\text{z-axis} \\ \text{alignment}}} \rightarrow \frac{-1}{2}\boldsymbol{I}_{i\pm}$$

where $\hat{\boldsymbol{u}}$ is a normalized vector $\hat{\boldsymbol{u}} = u_x\boldsymbol{i} + u_y\boldsymbol{j} + u_z\boldsymbol{k}$. If the tensor product ordering is reversed (using the function ***rev*** flag) there will be a sign change on all returned components.

See Also: T1, T2

10.8.8 T22SS

Usage:

```
#include <spin_T.h>
spin_T T22SS(spin_sys &sys, int spin, coord &pt, int m, int rev=0);
spin_T T22SS(spin_sys &sys, spin_op &Im, spin_op &Iz, spin_op &Ip, coord &pt, int m, int rev=0);
```

Description:

The function ***T22SS*** returns either the standard irreducible rank 2 space/spin tensor or a specific component of that tensor. The argument ***rev*** is not utilized, it is included for consistency with the function ***T21SS***.

1. T21SS(spin_sys &sys, int spin, coord &pt, int rev=0) - When *T2* is invoked with a spin system, ***sys***, a spin index, ***spin*** and a coordinate point, ***pt***, the function returns an irreducible spin-space spherical rank 2 tensor containing the five components of equation on page 192. This irreducible rank 2 tensor corresponds to a single spin because it is formulated from the product spin angular momentum vector (rank1 spin tensor) and a spatial vector (rank 1 spatial tensor) as given by the coordinate point. The coordinate is assumed to represent a normalized Cartesian vector. If the flag ***rev*** is set to non-zero, the tensor is assumed to be the product of the spatial vector times the spin vector as opposed to the spin vector times the spatial vector (default).
2. T21SS(spin_sys &sys, spin_op &Im, spin_op &Iz, spin_op &Ip, coord &pt, int rev=0) - This function performs the equivalent to 1. above except that the spin operators involved are directly input. This avoids any repetitive computation of the spin operators on successive calls to the function. ***Im*** is spin operator I_-, ***Iz*** is the spin operator I_z and ***Ip*** the spin operator I_+, all for the spin being treated.

The irreducible rank 2 space/spin tensor consists of five spherical components specified as

$$T^{(2)}_{2,m}(i)\ ,$$

where i is the spin index. The tensor index the tensor index ***m*** spans the rank, ***m*** ∈ ***[-2, 2]***. All tensor components are returned as spin operators.

Return Value:

An irreducible rank 2 space/spin tensor or a component of that tensor.

Example(s):

```
#include <gamma.h>
main()
  {
  spin_sys AMX(3);                    // create a three spin system called AMX.
  cout << SOp;                        // prints SOp, the 1,1 component of spin tensor.
  }
```

Mathematical Basis:

The five tensor components are given by the following equations,

Rank 2 Spin-Space Tensor Irreducible Spherical 2,m Components

$$\left\{ T^{(2)}_{2,0}(i) = \frac{1}{\sqrt{6}}[3\boldsymbol{I}_{iz}u_z - (\grave{\boldsymbol{I}}_i \bullet \grave{\boldsymbol{u}})] \right\}_{\substack{\text{z-axis} \\ \text{alignment}}} \rightarrow \frac{2}{\sqrt{6}}\boldsymbol{I}_{iz}$$

$$\left\{ T^{(2)}_{2,\pm 1}(i) = \mp\frac{1}{2}[\boldsymbol{I}_{i\pm}\ u_z + \boldsymbol{I}_{iz}u_{\pm}\ ] \right\}_{\substack{\text{z-axis} \\ \text{alignment}}} \rightarrow \mp\frac{1}{2}\boldsymbol{I}_{i\pm}$$

$$\left\{ T^{(2)}_{2,\pm 2}(i) = \frac{1}{2}[\boldsymbol{I}_{i\pm}\ u_{\pm}\ ] \right\}_{\substack{\text{z-axis} \\ \text{alignment}}} \rightarrow 0$$

where $\grave{\boldsymbol{u}}$ is a normalized vector $\grave{\boldsymbol{u}} = u_x\boldsymbol{i} + u_y\boldsymbol{j} + u_z\boldsymbol{k}$.

See Also: T1, T2

10.9 Complex Routines

10.9.1 component

Usage:

#include <spin_T.h>

spin_op space_T::component(int l, int m);

Description:

The member function *component* provides direct access to the individual components of the input tensor. A spin operator is returned which is

$$T^{(k)}_{l,\,m},$$

where ***k*** is the (implicit) rank of the tensor. The tensor components cannot lie outside of their intrinsic range as set by the rank ***k***. These are , $l \in [0, k]$ and $m \in [-l, l]$.

Return Value:

A spin operator is returned.

Example(s):

```
#include <gamma.h>
main()
  {
  spin_sys AX(2);                  // create a two spin system called AX.
  spin_op SOp;                     // new spin operator associated with spin system AX.
  spin_T SphT(AX);                 // new spin tensor associated with spin system AX.
  SphT = T2(AX, 0, 1);             // set SphT to rank 2 spin tensor for the AX system.
  SOp = SphT.component(2,-2);      // set SOp to the 2,-2 component of the tensor SphT.
  }
```

See Also:

10.9.2 rotate

Usage:

#include <spin_T.h>

spin_T spin_T::rotate = (double alpha, double beta, double gamma);

spin_T spin_T::rotate = (coord &EA);

Description:

The function ***rotate*** rotates the spin tensor coordinate system by Euler angles ***alpha***, ***beta***, and ***gamma*** to produce a new tensor.

1. rotate(double alpha, double beta, double gamma) - When ***rotate*** is invoked with a these arguments it rotates the coordinate system to which the tensor is referenced to by the Euler angles ***alpha***, ***beta***, and ***gamma***. The angles are input in degrees
2. rotate(coord &EA) - This use performs identically to 1. except that the three Euler angles ***alpha***, ***beta***, and ***gamma*** are contained in a single coordinate ***EA***.

Return Value:

A new spin tensor is returned.

Example(s):

```
#include <gamma.h>
main()
  {
  spin_sys AX(2);                    // create a two spin system called AX.
  spin_T SphT(AX);                   // new spin tensor associated with spin system AX.
  SphT = T2(AX, 0, 1);               // set SphT to the rank 2 spin tensor for AX system.
  SphT = SphT.rotate(0, 90.0, 0);    // rotate coordinate axes of SphT 90 degrees about y.
  coord EA(0, 90.0, 0);              // define a coordinate point with Euler angles.
  SphT = SphT.rotate(EA);            // another rotation of SphT 90 degrees about y.
  }
```

See Also: None

Mathematical Basis:

10.9.3 T_mult

Usage:

```
#include <spin_T.h>
spin_T T_mult = (spin_T &SphT1, spin_T &SphT2);
```

Description:

The function ***T_mult*** multiplies two spherical irreducible tensors together (cross product) to produce a new tensor, ***T = T1 x T2***. The resulting tensor will have a rank which is the sum of the individual tensor ranks and will be stored in terms of its irreducible spherical components.

Return Value:

A spin tensor is returned.

Example(s):

```
#include <gamma.h>
main()
  {
  spin_sys AX(2);                    // create a two spin system called AX.
  spin_T SphT(AX);                   // new spin tensor associated with spin system AX.
  spin_T SphT1(AX);                  // new spin tensor associated with spin system AX.
  spin_T SphT2(AX);                  // new spin tensor associated with spin system AX.
  SphT1 = T1(AX, 0);                 // SphT1 rank1 spin tensor for the AX system, spin 0.
  SphT2 = T1(AX, 1);                 // SphT2 rank1 spin tensor for the AX system, spin 1.
  SphT = T_prod(SphT1, SphT2);       // set SphT equal to the product SphT1 x SphT2
  }
```

See Also: None

Mathematical Basis:

The product of two irreducible spherical tensors produces another spherical irreducible tensor, ***T = T1 x T2***, whose components are given by equation (10-13)[1]

$$T_{L,M} = \sum_{m1}\sum_{m2}^{l1+l2} T1_{l1,m1} T2_{l2,m2} \langle l1\, l2\, m1\, m2 | LM \rangle$$

where $L \in [l1+l2, |l1-l2|]$, $M = m1 + m2$ and $\langle l1\, l2\, m1\, m2 | LM \rangle$ are Clebsch-Gordan coefficients. Since the Clebsch-Gordan coefficients are directly related to Wigner 3-j coefficients[1], the latter having more symmetry properties than the former, we can equally well write (10-13) with the Wigner 3-j symbols. This has been demonstrated in the discussion section at the end of this Chapter and produced equation (10-13)[2].

$$T_{L,M} = (-1)^{2l1+L-M}\sqrt{2L+1}\sum_{m1}\sum_{m2}^{l1+l2} T1_{l1,m1} T2_{l2,m2} \begin{pmatrix} L & l1 & l2 \\ M & -m1 & -m2 \end{pmatrix}$$

10.9.4 <<

Usage:

```
#include <spin_T.h>
ostream& << (ostream& ost, spin_T& SphT);
```

Description:

The operator << provides standard output abilities for spin tensors.

Return Value:

An ostream contains all components of the spin tensor specified.

Example(s):

```
#include <gamma.h>
main()
  {
  spin_sys ABX(3);                  // create a three spin system called ABX.
  spin_T SphT(ABX);                 // create an associated spin tensor called SphT.
  SphT = T_dip(ABX, 1, 2);          // SphT to dipolar tensor for last two spins of ABX.
  cout << SphT;                     // prints tensor, i.e. all 9 tensor components (SOps).
  }
```

1. This formula is found in Brink and Satchler, page 52, equation (4.6).
1. See the functions Wigner_3j and Clebsch_Gordan in this Chapter.
2. This equation corresponds to Brink and Satchler, page 123 in Appendix VI.

10.10Auxiliary Functions

10.10.1 Wigner_3j

Usage:

```
#include <spin_T.h>
double Wigner_3j(int a, int b, int c, int alpha, int beta, int gamma);
```

Description:

The function ***Wigner_3j*** returns the Wigner 3-j coefficient as a double precision number. The top row of the coefficient is specified by integers a, b, and c whereas the bottom row is specified by integers alpha, beta, and gamma.

$$\begin{pmatrix} a & b & c \\ \alpha & \beta & \gamma \end{pmatrix}$$

Return Value:

The Wigner 3-j coefficient is returned as a double precision number.

Example(s):

```
#include <gamma.h>
main()
  {
  for(int i=1; i<7; i++)                          // print first 6 coefficients in the ensuing Figure.
    cout << "\n\tElement 0" << i << i << ": " << Wigner_3j(0,i,i,0,0,0);
  }
```

Mathematical Basis:

The Wigner 3-j coefficients are computed from the Clebsch-Gordan coefficients via equation equation (10-33) on page 227[1]

$$\begin{pmatrix} a & b & c \\ \alpha & \beta & \gamma \end{pmatrix} = (-1)^{\gamma + b - a}\frac{1}{\sqrt{2c+1}}\langle ab\alpha\beta|c\text{-}\gamma\rangle$$

Some of the Wigner 3-j coefficients are tabulated here for use as a check of this function.

1. Brink and Satchler, page 39, equation (3.3). This formula is also found in their Appendix I, page 113.

These were generated, in part, from the example code previously given.

Values of $\begin{pmatrix} a & b & c \\ 0 & 0 & 0 \end{pmatrix}$

a b c	$\begin{pmatrix} a & b & c \\ 0 & 0 & 0 \end{pmatrix}$		a b c	$\begin{pmatrix} a & b & c \\ 0 & 0 & 0 \end{pmatrix}$		a b c	$\begin{pmatrix} a & b & c \\ 0 & 0 & 0 \end{pmatrix}$	
0 1 1	$-\sqrt{1/3}$	-0.577	1 5 6	$\sqrt{6/143}$	+0.205	3 3 6	$\sqrt{100/3003}$	+0.182
0 2 2	$\sqrt{1/5}$	+0.447	2 2 2	$-\sqrt{2/35}$	-0.239	3 4 5	$\sqrt{20/1001}$	+0.141
0 3 3	$-\sqrt{1/7}$	-0.378	2 2 4	$\sqrt{2/35}$	+0.239	3 5 6	$-\sqrt{7/429}$	-0.128
0 4 4	$\sqrt{1/9}$	+0.333	2 3 3	$\sqrt{4/105}$	+0.195	4 4 4	$\sqrt{18/1001}$	+0.134
0 5 5	$-\sqrt{1/11}$	-0.301	2 3 5	$-\sqrt{10/231}$	-0.208	4 4 6	$-\sqrt{20/1287}$	-0.125
0 6 6	$\sqrt{1/13}$	+0.277	2 4 4	$-\sqrt{20/693}$	-0.170	4 5 5	$-\sqrt{2/143}$	-0.118
1 1 2	$\sqrt{2/15}$	+0.365	2 4 6	$\sqrt{4/143}$	+0.187	4 6 6	$\sqrt{28/2431}$	+0.107
1 2 3	$-\sqrt{3/35}$	-0.293	2 5 5	$\sqrt{10/429}$	+0.153	5 5 6	$\sqrt{80/7293}$	+0.105
1 3 4	$\sqrt{4/63}$	+0.252	2 6 6	$-\sqrt{14/715}$	-0.140	6 6 6	$-\sqrt{400/46189}$	-0.093
1 4 5	$-\sqrt{5/99}$	-0.225	3 3 4	$-\sqrt{2/77}$	-0.161			

Table 16-1 ***Values[1] of selected Wigner 3-j coefficients with*** $\alpha = \beta = \gamma = 0$.

These can be verified by a direct computation using equation (10-34) on page 227 which applies when $\alpha = \beta = \gamma = 0$ [2].

$$\begin{pmatrix} a & b & c \\ 0 & 0 & 0 \end{pmatrix} = (-1)^g \Delta(abc) g! [(g-a)!(g-b)!(g-c)!]^{-1}$$

where $2g = a + b + c$ and

$$\Delta(abc) = \left[\frac{(a-b-c)!(a+c-b)!(b+c-a)!}{(a+b+c+1)!} \right]^{1/2} .$$

See Also: Clebsch_Gordan

1. Brink and Satchler, page 35, TABLE 2. Note that there is a sign error in their table on the 4 5 5 element as can be demonstrated from equation (10-34).
2. Brink and Satchler, page 34, equation (2.35) combined with the relationship between the Clebsch-Gordan coefficients and the Wigner 3-j symbols.

10.10.2 Clebsch_Gordan

Usage:

```
#include <spin_T.h>
double Clebsch_Gordan(int a, int b, int alpha, int beta, int c, int gamma);
```

Description:

The function *Clebsch_Gordan* returns the Clebsch-Gordan coefficient as a double precision number. The first four component indices are specified by integers ***a***, ***b***, α, and β, whereas the last two are specified by integers ***c***, and γ, $\langle ab\alpha\beta|c\gamma\rangle$

Return Value:

The Clebsch-Gordan coefficient is returned as a double precision number.

Example(s):

```
#include <gamma.h>
main()
  {
  double r;                              // create a two spin system called AX.
  r = Clebsch_Gordan(1,2,0,0,2,0);       // set r to <1200|20>.
  }
```

Mathematical Basis:

The Clebsch-Gordan coefficients are computed via equation equation (10-31) on page 227[1]

$$\langle ab\alpha\beta|c\gamma\rangle = \delta_{\alpha+\beta,\gamma}\times\Delta(abc)\times[(2c+1)(a+\alpha)!(a-\alpha)!(b+\beta)!(b-\beta)!(c+\gamma)!(c-\gamma)!]^{1/2}$$

$$\times\sum_{\nu}(-1)^{\nu}[(a-\alpha-\nu)!(c-b+\alpha+\nu)!(b+\beta-\nu)!(c-a-\beta+\nu)!\nu!(a+b-c-\gamma)!]^{-1}$$

where the function Δ***(abc)***is obtained from

$$\Delta(abc) = \left[\frac{(a-b-c)!(a+c-b)!(b+c-a)!}{(a+b+c+1)!}\right]^{1/2}$$

and the summation runs from ***v*** equal to zero through the positive integers as long as all factorials are non-negative.

See Also: Wigner_3j

1. A general formula for the Clebsch-Gordan coefficients is given in Brink and Satchler, page 34, equation (2.34). The relationship between the Clebsch-Gordan coefficients and Wigner 3-j coefficients is found on page 113 of their Appendix 1.

10.11 Description

10.11.1 Introduction

Class SPIN TENSOR (spin_T) contains routines to handle general tensors involving spin angular momentum. Each spin tensor is maintained in its irreducible spherical form, each component being a spin operator.

10.11.2 Generalized Hamiltonian

Hamiltonians can usually be expressed in terms of scalar products of tensors. The main advantage in doing so is to utilize the well defined rotational properties of tensors, thus allowing for rotations of entire Hamiltonians. To demonstrate the tensor formalism we consider the interactions of nuclear spins, either with other nuclear spins or with an applied magnetic field. In the first case, the Hamiltonian between two spins can be written

$$\boldsymbol{H}(i,j) = \grave{\boldsymbol{I}}_i \bullet \hat{\boldsymbol{A}}_{ij} \bullet \grave{\boldsymbol{I}}_j \,, \tag{18-1}$$

and in the latter, a spin interacting with an applied field $\boldsymbol{B}$, one may write

$$\boldsymbol{H}(i) = \grave{\boldsymbol{I}}_i \bullet \hat{\boldsymbol{A}}_i \bullet \grave{\boldsymbol{B}} \,. \tag{10-1}$$

In both examples $\grave{\boldsymbol{I}}_i$ is the spin angular momentum operator of spin i (a vector), and $\hat{\boldsymbol{A}}$ is a matrix containing information about the interaction under consideration - strength and geometry. In matrix form these equations looks like

$$\boldsymbol{H}(i,j) = \begin{bmatrix} \boldsymbol{I}_{ix} & \boldsymbol{I}_{iy} & \boldsymbol{I}_{iz} \end{bmatrix} \bullet \begin{bmatrix} A_{xx} & A_{xy} & A_{xz} \\ A_{yx} & A_{yy} & A_{yz} \\ A_{zx} & A_{zy} & A_{zz} \end{bmatrix}_{ij} \bullet \begin{bmatrix} \boldsymbol{I}_{jx} \\ \boldsymbol{I}_{jy} \\ \boldsymbol{I}_{jz} \end{bmatrix} \qquad \boldsymbol{H}(i) = \begin{bmatrix} \boldsymbol{I}_{ix} & \boldsymbol{I}_{iy} & \boldsymbol{I}_{iz} \end{bmatrix} \bullet \begin{bmatrix} A_{xx} & A_{xy} & A_{xz} \\ A_{yx} & A_{yy} & A_{yz} \\ A_{zx} & A_{zy} & A_{zz} \end{bmatrix}_{i} \bullet \begin{bmatrix} B_x \\ B_y \\ B_z \end{bmatrix} \tag{10-2}$$

which are compactly written as

$$\boldsymbol{H}(i) = \sum_{u}^{axes} \sum_{v}^{axes} \langle 1|\grave{\boldsymbol{I}}_i|u\rangle\langle u|\hat{\boldsymbol{A}}_{ij}|v\rangle\langle v|\grave{\boldsymbol{I}}_j|1\rangle \qquad \boldsymbol{H}(i) = \sum_{u}^{axes} \sum_{v}^{axes} \langle 1|\grave{\boldsymbol{I}}_i|u\rangle\langle u|\hat{\boldsymbol{A}}_i|v\rangle\langle v|\hat{\boldsymbol{B}}|1\rangle \tag{10-3}$$

with $\boldsymbol{u, v} \in \boldsymbol{\{x, y, z\}}$ and $\grave{\boldsymbol{B}}$ a magnetic field vector. These equations can be rearranged to produce an equation involving two rank 2 Cartesian tensors by taking the dyadic product of the two vectors involved. The two equation in this form are

$$\begin{aligned} \boldsymbol{H}(i,j) &= \sum_{u}^{axes} \sum_{v}^{axes} \langle u|\hat{\boldsymbol{A}}_{ij}|v\rangle\langle v|\grave{\boldsymbol{I}}_j|1\rangle\langle 1|\grave{\boldsymbol{I}}_i|u\rangle = \sum_{u}^{axes} \sum_{v}^{axes} \langle u|\hat{\boldsymbol{A}}_{ij}|v\rangle\langle v|\grave{\boldsymbol{I}}_j\grave{\boldsymbol{I}}_i|u\rangle \\ \boldsymbol{H}(i) &= \sum \sum \langle u|\hat{\boldsymbol{A}}_i|v\rangle\langle v|\grave{\boldsymbol{B}}|1\rangle\langle 1|\grave{\boldsymbol{I}}_i|u\rangle = \sum \sum \langle u|\hat{\boldsymbol{A}}_i|v\rangle\langle v|\grave{\boldsymbol{B}}\grave{\boldsymbol{I}}_i|u\rangle \end{aligned} \tag{10-4}$$

The dyadic product to produce $\bar{\bar{\boldsymbol{B}}}\grave{\bar{\boldsymbol{I}}}_i$ is explicitly done *via*

$$\begin{bmatrix} B_x \\ B_y \\ B_z \end{bmatrix} \bullet \begin{bmatrix} \boldsymbol{I}_{ix} & \boldsymbol{I}_{iy} & \boldsymbol{I}_{iz} \end{bmatrix} = \begin{bmatrix} B_x\boldsymbol{I}_{xi} & B_x\boldsymbol{I}_{yi} & B_x\boldsymbol{I}_{zi} \\ B_y\boldsymbol{I}_{xi} & B_y\boldsymbol{I}_{yi} & B_y\boldsymbol{I}_{zi} \\ B_z\boldsymbol{I}_{xi} & B_z\boldsymbol{I}_{yi} & B_z\boldsymbol{I}_{zi} \end{bmatrix}, \tag{10-5}$$

placing our two example treatments in the forms

$$\boldsymbol{H}(i,j) = \begin{bmatrix} A_{xx} & A_{xy} & A_{xz} \\ A_{yx} & A_{yy} & A_{yz} \\ A_{zx} & A_{zy} & A_{zz} \end{bmatrix}_{ij} \bullet \begin{bmatrix} \boldsymbol{I}_{xj}\boldsymbol{I}_{xi} & \boldsymbol{I}_{xj}\boldsymbol{I}_{yi} & \boldsymbol{I}_{xj}\boldsymbol{I}_{zi} \\ \boldsymbol{I}_{yj}\boldsymbol{I}_{xi} & \boldsymbol{I}_{yj}\boldsymbol{I}_{yi} & \boldsymbol{I}_{yj}\boldsymbol{I}_{zi} \\ \boldsymbol{I}_{zj}\boldsymbol{I}_{xi} & \boldsymbol{I}_{zj}\boldsymbol{I}_{yi} & \boldsymbol{I}_{zj}\boldsymbol{I}_{zi} \end{bmatrix} \qquad \boldsymbol{H}(i) = \begin{bmatrix} A_{xx} & A_{xy} & A_{xz} \\ A_{yx} & A_{yy} & A_{yz} \\ A_{zx} & A_{zy} & A_{zz} \end{bmatrix}_{i} \bullet \begin{bmatrix} B_x\boldsymbol{I}_{xi} & B_x\boldsymbol{I}_{yi} & B_x\boldsymbol{I}_{zi} \\ B_y\boldsymbol{I}_{xi} & B_y\boldsymbol{I}_{yi} & B_y\boldsymbol{I}_{zi} \\ B_z\boldsymbol{I}_{xi} & B_z\boldsymbol{I}_{yi} & B_z\boldsymbol{I}_{zi} \end{bmatrix} \tag{10-6}$$

An alternative treatment used in the latter case (single spin) is to rearrange (10-3) to produce an equation involving two rank 1 Cartesian tensors as opposed to rank 2 tensors.

$$\boldsymbol{H}(i) = \sum_{u}^{\text{axes}} \sum_{v}^{\text{axes}} \langle 1|\grave{\bar{\boldsymbol{I}}}_i|u\rangle\langle u|\hat{\boldsymbol{A}}_i|v\rangle\langle v|\bar{\bar{\boldsymbol{B}}}|1\rangle = \sum_{u}^{\text{axes}} \langle 1|\grave{\bar{\boldsymbol{I}}}_i|u\rangle\langle u|\hat{\boldsymbol{A}}_i\bar{\bar{\boldsymbol{B}}}|1\rangle \tag{10-7}$$

The matrix form in the case is the product of two vectors.

$$\boldsymbol{H}(i) = \begin{bmatrix} \boldsymbol{I}_{ix} & \boldsymbol{I}_{iy} & \boldsymbol{I}_{iz} \end{bmatrix} \bullet \begin{bmatrix} A_{xx}B_x + A_{xy}B_y + A_{xz}B_z \\ A_{yx}B_x + A_{yy}B_y + A_{yz}B_z \\ A_{zx}B_x + A_{zy}B_y + A_{zz}B_z \end{bmatrix}. \tag{10-8}$$

This equation, along with (10-6), has a generalized Hamiltonian written in terms of two Cartesian tensors, one involving only spatial coordinates $\boldsymbol{A}$ and one involving only spin coordinates $\boldsymbol{T}$. Using more direct tensor notation, we have from (10-6) and (10-8)

$$\boldsymbol{H}(i,j) = \boldsymbol{A}_{ij}^{(2)} \bullet \boldsymbol{T}_{ij}^{(2)} \qquad \boldsymbol{H}(i) = \boldsymbol{A}_{i}^{(1)} \bullet \boldsymbol{T}_{i}^{(1)} \tag{10-9}$$

The separation of the Hamiltonian into spatial and spin tensor components can allow for facile rotations of $\boldsymbol{H}$ in either coordinate or spin space. Similar notation is also utilized to describe equation sosi

$$\boldsymbol{H}(i) = \boldsymbol{A}_{i}^{(2)} \bullet \boldsymbol{T}_{i}^{(2)} \ . \tag{10-10}$$

but in this instance there are spatial components in both tensors. Thus one must then be careful when performing rotations in this particular case[1].

Now we address how one can rotate these Hamiltonians. In order to do so we must once again rewrite the Hamiltonian in terms of components which behave in a well defined manner under a rotation. For this we need the tensors in irreducible spherical format.

1. This applies to the treatment of the chemical shift Hamiltonian in NMR. In that instance the Hamiltonian is rotated by keeping the space/spin tensor aligned along the z-axis in the laboratory frame (the axis of the static spectrometer magnetic field) and re-orienting the pure spatial tensor.

10.11.3 Ladder Operator Relationship

An irreducible tensor can be defined in terms of commutation rules involving its components and angular momentum operators. The set of $\boldsymbol{T_{l,m}}$ components constitute an irreducible tensor if the following two commutation relationships are fulfilled.

$$[\boldsymbol{J}_z, \boldsymbol{T}_{l,m}] = m\boldsymbol{T}_{l,m} \qquad [\boldsymbol{J}_\pm, \boldsymbol{T}_{l,m}] = \sqrt{(l \pm m + 1)(l \mp m)}\,\boldsymbol{T}_{l,m\pm 1} \tag{10-11}$$

Given a single tensor component all other components can be generated with these two equations.

10.11.4 Rank 1 Spherical Tensors via Ladder Operators

Consider a rank ***1***, ***l=1***, irreducible tensor[1]. By choosing a single component we can determine the two others. From $\boldsymbol{[J_z, T_{l,m}] = mT_{l,m}}$ an obvious choice for $\boldsymbol{T_{1,0}}$ would be $\boldsymbol{J_z}$ itself since it corresponds to the ***m=0***. If we set $\boldsymbol{T_{1,0} = CJ_z}$ where ***C*** is an arbitrary constant, the left equation (10-11) will satisfied and we can derive the other two rank 1 components from the other commutation relationship, namely the right equation (10-11), in the following manner.

$$[\boldsymbol{J}_\pm, \boldsymbol{T}_{1,0}] = \sqrt{(1 \pm 0 + 1)(1 \mp 0)}\,\boldsymbol{T}_{1,0\pm 1} = \sqrt{2}\boldsymbol{T}_{1,\pm 1}$$

$$[\boldsymbol{J}_\pm, \boldsymbol{T}_{1,0}] = [\boldsymbol{J}_\pm, C\boldsymbol{J}_z] = -C[\boldsymbol{J}_z, \boldsymbol{J}_\pm] = \mp C\boldsymbol{J}_\pm$$

$$\sqrt{2}\boldsymbol{T}_{1,\pm 1} = \mp C\boldsymbol{J}_\pm \rightarrow \boldsymbol{T}_{1,\pm 1} = \mp C\frac{1}{\sqrt{2}}\boldsymbol{J}_\pm$$

Apparently a valid rank 1 irreducible tensor could be[2] (arbitrary overall scaling)

$$\boldsymbol{T}_1: \qquad \left\{ -\boldsymbol{T}_{1,-1} = \frac{1}{\sqrt{2}}\boldsymbol{J}_+ \qquad \boldsymbol{T}_{1,0} = \boldsymbol{J}_z \qquad \boldsymbol{T}_{1,1} = -\frac{1}{\sqrt{2}}\boldsymbol{J}_- \right\} \tag{10-12}$$

Throughout GAMMA we keep this scaling and all rank 1 spin tensors generated have the form.

$$T_{1,0}(i) = I_{iz} \qquad T_{1,\pm 1}(i) = \frac{\mp 1}{\sqrt{2}} I_{i\pm}$$

Figure A Rank 1 irreducible spherical spin tensor components. Index i indicates a single particle with spin angular momentum & I it's associated operator.

1. There is no rank 0 tensor, $T_{0,0}(i) = 0$

2. The derivation to obtain these forms utilized the identity $[\boldsymbol{J}_z, \boldsymbol{J}_\pm] = \pm\boldsymbol{J}_\pm$ or equivalently $[\boldsymbol{J}_\pm, \boldsymbol{J}_z] = \mp\boldsymbol{J}_\pm$

Rank 1 Irreducible Spherical Spin Tensor Components - Matrix Representations

$$T^{(1)}_{0,0} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \qquad T^{(1)}_{1,0} = \frac{1}{2}\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \qquad T^{(1)}_{1,-1} = \frac{1}{\sqrt{2}}\begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix} \qquad T^{(1)}_{1,1} = \frac{-1}{\sqrt{2}}\begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}$$

Figure B Matrix representations in Hilbert space for a single spin (I=1/2) of the rank 1 irreducible spherical spin tensor components.

10.11.5 Rank 2 Spherical Tensors via Ladder Operators

One can equally well derive all five components of the irreducible rank 2 tensor from a suitable initial choice of any one of the five. Starting from a suitable choice for the $\boldsymbol{T_{2,-2}}$ component, the commutations relationships of (10-11) will be used to derive the other four components. Consider the $\boldsymbol{J_z}$ commutation equation $[\boldsymbol{J}_z, \boldsymbol{T}_{l,m}] = m\boldsymbol{T}_{l,m}$ with $\boldsymbol{m = -2}$, $[\boldsymbol{J}_z, \boldsymbol{T}_{2,-2}] = (-2)\boldsymbol{T}_{2,-2}$ .Since[1] $[\boldsymbol{J}_z, \boldsymbol{J}_\pm] = \pm\boldsymbol{J}_\pm$ and $\boldsymbol{J_z = J_{iz}J_{jz}}$ it is not too difficult to surmise that $\boldsymbol{J_{i-}J_{j-}}$ will be a satisfactory choice. Then we begin by taking

$$\boldsymbol{T}_{2,-2} = C\boldsymbol{J}_{i-}\boldsymbol{J}_{j-}$$

where again $\boldsymbol{C}$ is any arbitrary constant. The derivation of $\boldsymbol{T_{2,-1}}$ from $\boldsymbol{T_{2,-2}}$ is relatively simple.

$$\begin{aligned}
[\boldsymbol{J}_\pm\ ,\ \boldsymbol{T}_{l,m}] &= \sqrt{(l \pm m + 1)(l \mp m)}\,\boldsymbol{T}_{l,m\pm 1} \\
[\boldsymbol{J}_\pm\ ,\ \boldsymbol{T}_{2,-2}] &= \sqrt{(2 + (-2) + 1)(2 - (-2))}\,\boldsymbol{T}_{2,-2+1} \\
[\boldsymbol{J}_{i+} + \boldsymbol{J}_{j+},\ C\boldsymbol{J}_{i-}\boldsymbol{J}_{j-}] &= 2\boldsymbol{T}_{2,-1} \\
\boldsymbol{T}_{2,-1} &= \frac{1}{2}[\boldsymbol{J}_{i+} + \boldsymbol{J}_{j+},\ C\boldsymbol{J}_{i-}\boldsymbol{J}_{j-}] = \frac{C}{2}([\boldsymbol{J}_{i+},\ \boldsymbol{J}_{i-}\boldsymbol{J}_{j-}] + [\boldsymbol{J}_{j+},\ \boldsymbol{J}_{i-}\boldsymbol{J}_{j-}])
\end{aligned}$$

Using the commutation relationship that $[\boldsymbol{J}_\pm\ ,\ \boldsymbol{J}_\mp] = \pm 2\boldsymbol{J}_z$ this becomes

$$\boldsymbol{T}_{2,-1} = \frac{C}{2}(2\boldsymbol{J}_{iz}\boldsymbol{J}_{j-} + 2\boldsymbol{J}_{i-}\boldsymbol{J}_{jz}) = C(\boldsymbol{J}_{iz}\boldsymbol{J}_{j-} + \boldsymbol{J}_{i-}\boldsymbol{J}_{jz})$$

This procedure will now be repeated to generate $\boldsymbol{T_{2,-0}}$ through $\boldsymbol{T_{2,-2}}$

$$\begin{aligned}
[\boldsymbol{J}_\pm\ ,\ \boldsymbol{T}_{l,m}] &= \sqrt{(l \pm m + 1)(l \mp m)}\,\boldsymbol{T}_{l,m\pm 1} \\
[\boldsymbol{J}_\pm\ ,\ \boldsymbol{T}_{2,-1}] &= \sqrt{(2 + (-1) + 1)(2 - (-1))}\,\boldsymbol{T}_{2,-1+1} \\
[\boldsymbol{J}_{i+} + \boldsymbol{J}_{j+},\ C(\boldsymbol{J}_{iz}\boldsymbol{J}_{j-} + \boldsymbol{J}_{i-}\boldsymbol{J}_{jz})] &= \sqrt{6}\,\boldsymbol{T}_{2,0}
\end{aligned}$$

1. Here the angular momentum is a sum from components i and j. The tensor themselves could equally well be indexed with i and j but this is not included here as the nomenclature gets very messy.

$$\boldsymbol{T}_{2,0} = \frac{C}{\sqrt{6}}\{[\boldsymbol{J}_{i+}, (\boldsymbol{J}_{iz}\boldsymbol{J}_{j-} + \boldsymbol{J}_{i-}\boldsymbol{J}_{jz})] + [\boldsymbol{J}_{j+}, (\boldsymbol{J}_{iz}\boldsymbol{J}_{j-} + \boldsymbol{J}_{i-}\boldsymbol{J}_{jz})]\}$$

$$= \frac{C}{\sqrt{6}}\{[\boldsymbol{J}_{i+}, \boldsymbol{J}_{iz}\boldsymbol{J}_{j-}] + [\boldsymbol{J}_{i+}, \boldsymbol{J}_{i-}\boldsymbol{J}_{jz}] + [\boldsymbol{J}_{j+}, \boldsymbol{J}_{iz}\boldsymbol{J}_{j-}] + [\boldsymbol{J}_{j+}, \boldsymbol{J}_{i-}\boldsymbol{J}_{jz}]\}$$

$$= \frac{C}{\sqrt{6}}\{[\boldsymbol{J}_{i+}, \boldsymbol{J}_{iz}]\boldsymbol{J}_{j-} + [\boldsymbol{J}_{i+}, \boldsymbol{J}_{i-}]\boldsymbol{J}_{jz} + \boldsymbol{J}_{iz}[\boldsymbol{J}_{j+}, \boldsymbol{J}_{j-}] + \boldsymbol{J}_{i-}[\boldsymbol{J}_{j+}, \boldsymbol{J}_{jz}]\}$$

Using the relationship that

$$[\boldsymbol{J}_{\pm}, \boldsymbol{J}_{\mp}] = \pm 2\boldsymbol{J}_z \qquad \text{and} \qquad [\boldsymbol{J}_{\pm}, \boldsymbol{J}_z] = \mp\boldsymbol{J}_{\pm}$$

this becomes

$$\boldsymbol{T}_{2,0} = \frac{C}{\sqrt{6}}(-\boldsymbol{J}_{i+}\boldsymbol{J}_{j-} + 2\boldsymbol{J}_{iz}\boldsymbol{J}_{jz} + \boldsymbol{J}_{iz}2\boldsymbol{J}_{jz} - \boldsymbol{J}_{i-}\boldsymbol{J}_{j+})$$

$$\boldsymbol{T}_{2,0} = \frac{C}{\sqrt{6}}(4\boldsymbol{J}_{iz}\boldsymbol{J}_{jz} - \boldsymbol{J}_{i+}\boldsymbol{J}_{j-} - \boldsymbol{J}_{i-}\boldsymbol{J}_{j+})$$

Repeating for the $\boldsymbol{T_{2,1}}$ component,

$$[\boldsymbol{J}_{\pm}, \boldsymbol{T}_{l,m}] = \sqrt{(l \pm m + 1)(l \mp m)}\,\boldsymbol{T}_{l,m\pm 1}$$

$$[\boldsymbol{J}_{\pm}, \boldsymbol{T}_{2,0}] = \sqrt{(2 + (0) + 1)(2 - (0))}\,\boldsymbol{T}_{2,0+1}$$

$$[\boldsymbol{J}_{i+} + \boldsymbol{J}_{j+}, \frac{C}{\sqrt{6}}(4\boldsymbol{J}_{iz}\boldsymbol{J}_{jz} - \boldsymbol{J}_{i+}\boldsymbol{J}_{j-} - \boldsymbol{J}_{i-}\boldsymbol{J}_{j+})] = \sqrt{6}\boldsymbol{T}_{2,1}$$

$$\boldsymbol{T}_{2,1} = \frac{C}{6}\{[\boldsymbol{J}_{i+}, (4\boldsymbol{J}_{iz}\boldsymbol{J}_{jz} - \boldsymbol{J}_{i+}\boldsymbol{J}_{j-} - \boldsymbol{J}_{i-}\boldsymbol{J}_{j+})] + [\boldsymbol{J}_{j+}, (4\boldsymbol{J}_{iz}\boldsymbol{J}_{jz} - \boldsymbol{J}_{i+}\boldsymbol{J}_{j-} - \boldsymbol{J}_{i-}\boldsymbol{J}_{j+})]\}$$

$$= \frac{C}{6}\{[\boldsymbol{J}_{i+}, 4\boldsymbol{J}_{iz}\boldsymbol{J}_{jz}] - [\boldsymbol{J}_{i+}, \boldsymbol{J}_{i+}\boldsymbol{J}_{j-}] - [\boldsymbol{J}_{i+}, \boldsymbol{J}_{i-}\boldsymbol{J}_{j+}]\}$$

$$+ \frac{C}{6}\{[\boldsymbol{J}_{j+}, 4\boldsymbol{J}_{iz}\boldsymbol{J}_{jz}] - [\boldsymbol{J}_{j+}, \boldsymbol{J}_{i+}\boldsymbol{J}_{j-}] - [\boldsymbol{J}_{j+}, \boldsymbol{J}_{i-}\boldsymbol{J}_{j+}]\}$$

$$= \frac{C}{6}\{4[\boldsymbol{J}_{i+}, \boldsymbol{J}_{iz}]\boldsymbol{J}_{jz} - [\boldsymbol{J}_{i+}, \boldsymbol{J}_{i+}]\boldsymbol{J}_{j-} - [\boldsymbol{J}_{i+}, \boldsymbol{J}_{i-}]\boldsymbol{J}_{j+} - 4\boldsymbol{J}_{iz}\boldsymbol{J}_{j+} - 2\boldsymbol{J}_{i+}\boldsymbol{J}_{jz} - \boldsymbol{J}_{i-}0\}$$

Using the two commutation relationships $[\boldsymbol{J}_{\pm}, \boldsymbol{J}_{\mp}] = \pm 2\boldsymbol{J}_z$ and $[\boldsymbol{J}_{\pm}, \boldsymbol{J}_z] = \mp\boldsymbol{J}_{\pm}$ this becomes

$$\boldsymbol{T}_{2,1} = \frac{C}{6}\{-4\boldsymbol{J}_{i+}\boldsymbol{J}_{jz} - 0\boldsymbol{J}_{j-} - 2\boldsymbol{J}_{iz}\boldsymbol{J}_{j+} - 4\boldsymbol{J}_{iz}\boldsymbol{J}_{j+} - 2\boldsymbol{J}_{i+}\boldsymbol{J}_{jz} - \boldsymbol{J}_{i-}0\}$$

and the final result

$$\boldsymbol{T}_{2,1} = -C(\boldsymbol{J}_{i+}\boldsymbol{J}_{jz} + \boldsymbol{J}_{iz}\boldsymbol{J}_{j+})$$

Finally for the $\boldsymbol{T_{2,2}}$ component, we first apply the ladder operator to $\boldsymbol{T_{2,1}}$

$$[\boldsymbol{J}_{\pm}\ ,\ \boldsymbol{T}_{l,\,m}] \;=\; \sqrt{(l \pm m + 1)(l \mp m)}\,\boldsymbol{T}_{l,\,m\pm 1}$$

$$[\boldsymbol{J}_{\pm}\ ,\ \boldsymbol{T}_{2,\,1}] \;=\; \sqrt{(2 + (1) + 1)(2 - (1))}\,\boldsymbol{T}_{2,\,1+1}$$

$$[\boldsymbol{J}_{i+} + \boldsymbol{J}_{j+}, -C(\boldsymbol{J}_{i+}\boldsymbol{J}_{jz} + \boldsymbol{J}_{iz}\boldsymbol{J}_{j+})] \;=\; 2\boldsymbol{T}_{2,\,2}$$

and next expand out the angular momentum operators

$$\boldsymbol{T}_{2,\,2} \;=\; \frac{-C}{2}\{[\boldsymbol{J}_{i+},\,(\boldsymbol{J}_{i+}\boldsymbol{J}_{jz} + \boldsymbol{J}_{iz}\boldsymbol{J}_{j+})] + [\boldsymbol{J}_{j+},\,(\boldsymbol{J}_{i+}\boldsymbol{J}_{jz} + \boldsymbol{J}_{iz}\boldsymbol{J}_{j+})]\}$$

$$= \frac{-C}{2}\{[\boldsymbol{J}_{i+},\,\boldsymbol{J}_{i+}\boldsymbol{J}_{jz}] + [\boldsymbol{J}_{i+},\,\boldsymbol{J}_{iz}\boldsymbol{J}_{j+}] + [\boldsymbol{J}_{j+},\,\boldsymbol{J}_{i+}\boldsymbol{J}_{jz}] + [\boldsymbol{J}_{j+},\,\boldsymbol{J}_{iz}\boldsymbol{J}_{j+}]\}$$

$$= \frac{-C}{2}\{[\boldsymbol{J}_{i+},\,\boldsymbol{J}_{i+}]\boldsymbol{J}_{jz} + [\boldsymbol{J}_{i+},\,\boldsymbol{J}_{iz}]\boldsymbol{J}_{j+} + \boldsymbol{J}_{i+}[\boldsymbol{J}_{j+},\,\boldsymbol{J}_{jz}] + \boldsymbol{J}_{iz}[\boldsymbol{J}_{j+},\,\boldsymbol{J}_{j+}]\}$$

Using the commutation relationship $[\boldsymbol{J}_{\pm}\ ,\ \boldsymbol{J}_{z}] \;=\; \mp\boldsymbol{J}_{\pm}$ produces the final result.

$$\boldsymbol{T}_{2,\,2} \;=\; \frac{-C}{2}(0\boldsymbol{J}_{jz} - \boldsymbol{J}_{i+}\boldsymbol{J}_{j+} - \boldsymbol{J}_{i+}\boldsymbol{J}_{j+} + \boldsymbol{J}_{iz}0) \;=\; C\boldsymbol{J}_{i+}\boldsymbol{J}_{j+}$$

Based on the rank 1 irreducible tensor components given in equation (10-12) - the reason is shown in the section on tensor products which follows in the Chapter - we shall set the constant ***C*** to be ***1/2***. All of the rank two components are now collected together here for convenience.

Rank Two Irreducible Spherical Spin Tensor Components

$$\boldsymbol{T}_{2,\,0} \;=\; \frac{1}{2\sqrt{6}}(4\boldsymbol{J}_{iz}\boldsymbol{J}_{jz} - \boldsymbol{J}_{i+}\boldsymbol{J}_{j-} - \boldsymbol{J}_{i-}\boldsymbol{J}_{j+}) \;=\; \frac{1}{\sqrt{6}}(3\boldsymbol{J}_{iz}\boldsymbol{J}_{jz} - \boldsymbol{J}_{i}\bullet\boldsymbol{J}_{j})$$

$$\boldsymbol{T}_{2,\,1} \;=\; -\frac{1}{2}(\boldsymbol{J}_{i+}\boldsymbol{J}_{jz} + \boldsymbol{J}_{iz}\boldsymbol{J}_{j+}) \qquad \boldsymbol{T}_{2,\,-1} \;=\; \frac{1}{2}(\boldsymbol{J}_{iz}\boldsymbol{J}_{j-} + \boldsymbol{J}_{i-}\boldsymbol{J}_{jz})$$

$$\boldsymbol{T}_{2,\,-2} \;=\; \frac{1}{2}\boldsymbol{J}_{i-}\boldsymbol{J}_{j-} \qquad \boldsymbol{T}_{2,\,2} \;=\; \frac{1}{2}\boldsymbol{J}_{i+}\boldsymbol{J}_{j+}$$

Figure C General formulae for the rank 2 irreducible spherical spin tensor components.

The second form of the $\boldsymbol{T}_{2,0}$ component is found often in the literature. These are shown to be equivalent in the following.

$$\frac{1}{2\sqrt{6}}(4\boldsymbol{J}_{iz}\boldsymbol{J}_{jz} - \boldsymbol{J}_{i+}\boldsymbol{J}_{j-} - \boldsymbol{J}_{i-}\boldsymbol{J}_{j+}) \;=\; \frac{2}{2\sqrt{6}}\left(2\boldsymbol{J}_{iz}\boldsymbol{J}_{jz} - \frac{1}{2}[\boldsymbol{J}_{i+}\boldsymbol{J}_{j-} + \boldsymbol{J}_{i-}\boldsymbol{J}_{j+}]\right) \;=\; \frac{1}{\sqrt{6}}(2\boldsymbol{J}_{iz}\boldsymbol{J}_{jz} - [\boldsymbol{J}_{ix}\boldsymbol{J}_{jx} + \boldsymbol{J}_{iy}\boldsymbol{J}_{jy}])$$

$$= \frac{1}{\sqrt{6}}(3\boldsymbol{J}_{iz}\boldsymbol{J}_{jz} - [\boldsymbol{J}_{ix}\boldsymbol{J}_{jx} + \boldsymbol{J}_{iy}\boldsymbol{J}_{jy} + \boldsymbol{J}_{iz}\boldsymbol{J}_{jz}]) \;=\; \frac{1}{\sqrt{6}}(3\boldsymbol{J}_{iz}\boldsymbol{J}_{jz} - \boldsymbol{J}_{i}\bullet\boldsymbol{J}_{j})$$

Another useful property of these tensor components is

$$\boldsymbol{T}_{2,-m} = (-1)^m \boldsymbol{T}_{2,m}^{\dagger}$$

10.11.6Tensor Products

The product of two irreducible spherical tensors produces another tensor, ***T = T1 x T2***, whose spherical irreducible components are given by[1]

$$T_{L,M} = \sum_{m1}\sum_{m2}^{+l1+l2} T1_{l1,m1} T2_{l2,m2} \langle l1 l2 m1 m2 | LM \rangle \qquad (10\text{-}13)$$

where $L \in [l1+l2, |l1-l2|]$, $M = m1 + m2$ and $\langle l1 l2 m1 m2 | LM \rangle$ are Clebsch-Gordan coefficients. As depicted in the following diagram, the two irreducible tensors combine to form a reducible tensor with rank equal to the sum of the ranks of two tensor which formed it. This product can be broken up into irreducible tensors of its rank down to rank zero.

Tensor Products

IRREDUCIBLE $\mathbf{T}_l$: $\mathbf{T}_{lm}$		REDUCIBLE	IRREDUCIBLE $\mathbf{T}_L$: $\mathbf{T}_{LM}$
$\mathbf{T}_0$: $\mathbf{T}_{00}$ $\mathbf{T}_0$: $\mathbf{T}_{00}$	⊗→		$\mathbf{T}_0$: $\mathbf{T}_{00}$
$\mathbf{T}_0$: $\mathbf{T}_{00}$ $\mathbf{T}_1$: $\mathbf{T}_{11}$ $\mathbf{T}_{10}$ $\mathbf{T}_{1-1}$	⊗→		$\mathbf{T}_0$: None $\mathbf{T}_1$: $\mathbf{T}_{11}$ $\mathbf{T}_{10}$ $\mathbf{T}_{1-1}$
$\mathbf{T}_1$: $\mathbf{T}_{11}$ $\mathbf{T}_{10}$ $\mathbf{T}_{1-1}$ $\mathbf{T}_1$: $\mathbf{T}_{11}$ $\mathbf{T}_{10}$ $\mathbf{T}_{1-1}$	⊗→	$\mathbf{T}_2$	$\mathbf{T}_0$: $\mathbf{T}_{00}$ $\mathbf{T}_1$: $\mathbf{T}_{11}$ $\mathbf{T}_{10}$ $\mathbf{T}_{1-1}$ $\mathbf{T}_2$: $\mathbf{T}_{22}$ $\mathbf{T}_{21}$ $\mathbf{T}_{20}$ $\mathbf{T}_{2-1}$ $\mathbf{T}_{2-2}$
$\mathbf{T}_1$: $\mathbf{T}_{11}$ $\mathbf{T}_{10}$ $\mathbf{T}_{1-1}$ $\mathbf{T}_2$: $\mathbf{T}_{22}$ $\mathbf{T}_{21}$ $\mathbf{T}_{20}$ $\mathbf{T}_{2-1}$ $\mathbf{T}_{2-2}$	⊗→	$\mathbf{T}_3$	$\mathbf{T}_0$: None $\mathbf{T}_1$: $\mathbf{T}_{11}$ $\mathbf{T}_{10}$ $\mathbf{T}_{1-1}$ $\mathbf{T}_2$: $\mathbf{T}_{22}$ $\mathbf{T}_{21}$ $\mathbf{T}_{20}$ $\mathbf{T}_{2-1}$ $\mathbf{T}_{2-2}$ $\mathbf{T}_3$: $\mathbf{T}_{33}$ $\mathbf{T}_{32}$ $\mathbf{T}_{31}$ $\mathbf{T}_{30}$ $\mathbf{T}_{3-1}$ $\mathbf{T}_{3-2}$ $\mathbf{T}_{3-3}$

Figure D : Irreducible components resulting from selected Tensor Products.

The components of these irreducible tensors are obtained from the formula presented, equation (10-13). Since the Clebsch-Gordan coefficients are directly related to Wigner 3-j coefficients[2], the latter having more symmetry properties than the former, we can equally well write (10-13) with the Wigner 3-j symbols. Substitution of

$$\langle l1 l2 m1 m2 | L - M \rangle = (-1)^{l1 - l2 - M} (2L+1)^{1/2} \begin{pmatrix} l1 & l2 & L \\ m1 & m2 & M \end{pmatrix}, \qquad (10\text{-}14)$$

1. This formula is found in Brink and Satchler, page 52, equation (4.6).
2. See the functions Wigner_3j and Clebsch_Gordan in this chapter.

the inverse of equation (10-33), into equation (10-13) produces

$$T_{L,M} = (-1)^{l1-l2+M}\sqrt{2L+1}\sum_{m1}^{\pm l1}\sum_{m2}^{\pm l2} T1_{l1,m1}T2_{l2,m2}\begin{pmatrix} l1 & l2 & L \\ m1 & m2 & -M \end{pmatrix}, \tag{10-15}$$

and utilizing the symmetry properties of the 3-j coefficients this is equivalent to[1]

$$T_{L,M} = (-1)^{2l1+L-M}\sqrt{2L+1}\sum_{m1}^{\pm l1}\sum_{m2}^{\pm l2} T1_{l1,m1}T2_{l2,m2}\begin{pmatrix} L & l1 & l2 \\ M & -m1 & -m2 \end{pmatrix}. \tag{10-16}$$

10.11.7 Rank 2 From Rank 1 Tensor Product

To demonstrate the mathematics, consider the buildup of a rank 2 spin tensor from the product of two rank 1 spin tensors, in this case for spin angular momentum of spins i and j respectively. According to our formula this would be

$$\boldsymbol{T}_{L,M}(i,j) = (-1)^{2+L-M}\sqrt{2L+1}\sum_{m1}^{\pm 1}\sum_{m2}^{\pm 1}\boldsymbol{T}1_{1,m1}(i)\boldsymbol{T}2_{1,m2}(j)\begin{pmatrix} L & 1 & 1 \\ M & -m1 & -m2 \end{pmatrix} \tag{10-17}$$

The components of the first irreducible rank 1 tensor in this case are (See equation sosi.)

$$T_{1,0}(i) = \boldsymbol{I}_{iz} \quad \& \quad T_{1,\pm 1}(i) = \mp\frac{1}{\sqrt{2}}[\boldsymbol{I}_{ix} \pm i\boldsymbol{I}_{iy}] = \mp\frac{1}{\sqrt{2}}\boldsymbol{I}_{i\pm} \quad , \tag{10-18}$$

and the components of the second rank 1 tensor the same with the spin index switched from i to j.

0.0.5.1 GENERATION OF $T_{0,0}(i,j)$

Proceeding to build up the components of the resultant rank 2 tensor, for the 0,0 component we have from equation (10-17)

$$\boldsymbol{T}_{0,0}(i,j) = (-1)^{2+0-0}\sqrt{2(0)+1}\sum_{m1}^{\pm 1}\sum_{m2}^{\pm 1}\boldsymbol{T}1_{1,m1}(i)\boldsymbol{T}2_{1,m2}(j)\begin{pmatrix} 0 & 1 & 1 \\ 0 & -m1 & -m2 \end{pmatrix}$$

$$\boldsymbol{T}_{0,0}(i,j) = \boldsymbol{T}1_{1,0}(i)\boldsymbol{T}2_{1,0}(j)\begin{pmatrix} 0 & 1 & 1 \\ 0 & 0 & 0 \end{pmatrix} + \boldsymbol{T}1_{1,1}(i)\boldsymbol{T}2_{1,-1}(j)\begin{pmatrix} 0 & 1 & 1 \\ 0 & -1 & 1 \end{pmatrix} + \boldsymbol{T}1_{1,-1}(i)\boldsymbol{T}2_{1,1}(j)\begin{pmatrix} 0 & 1 & 1 \\ 0 & 1 & -1 \end{pmatrix} \tag{10-19}$$

The rank one tensors are given in equation (10-18) and the three Wigner 3-j coefficients are obtainable from equations sosi, sosi, and sosi.

1. This equation corresponds to Brink and Satchler, page 123 in Appendix VI.

$$\begin{pmatrix} a & a+1 & 1 \\ \alpha & -\alpha & 0 \end{pmatrix} = (-1)^{a-\alpha-1}\sqrt{\frac{(a-\alpha+1)(a+\alpha+1)}{(a+1)(2a+1)(2a+3)}} \rightarrow \begin{pmatrix} 0 & 1 & 1 \\ 0 & 0 & 0 \end{pmatrix} = (-1)^{-1}\sqrt{\frac{(1)(1)}{(1)(1)(3)}} = -\frac{1}{\sqrt{3}}$$

$$\begin{pmatrix} a & a+1 & 1 \\ \alpha & -\alpha-1 & 1 \end{pmatrix} = (-1)^{a-\alpha}\sqrt{\frac{(a+\alpha+1)(a+\alpha+2)}{(2a+1)(2a+2)(2a+3)}} \rightarrow \begin{pmatrix} 0 & 1 & 1 \\ 0 & -1 & 1 \end{pmatrix} = (-1)^{0}\sqrt{\frac{(1)(2)}{(1)(2)(3)}} = \frac{1}{\sqrt{3}}$$

From the previous result and sosi,

$$\begin{pmatrix} 0 & 1 & 1 \\ 0 & 1 & -1 \end{pmatrix} = (-1)^{0+1+1}\begin{pmatrix} 0 & 1 & 1 \\ 0 & -1 & 1 \end{pmatrix} = \frac{1}{\sqrt{3}} \quad .$$

Placing these coefficients and the rank 1 components of (10-18) into equation (10-19) produces

$$\boldsymbol{T}_{0,0}(i,j) = \left(-\frac{1}{\sqrt{3}}\right)(\boldsymbol{I}_{iz})(\boldsymbol{I}_{jz}) + \frac{1}{\sqrt{3}}\left[-\frac{1}{\sqrt{2}}\boldsymbol{I}_{i+}\right]\left[\frac{1}{\sqrt{2}}\boldsymbol{I}_{j-}\right] + \frac{1}{\sqrt{3}}\left[\frac{1}{\sqrt{2}}\boldsymbol{I}_{i-}\right]\left[-\frac{1}{\sqrt{2}}\boldsymbol{I}_{j+}\right]$$

$$\boxed{\boldsymbol{T}_{0,0}(i,j) = -\frac{1}{\sqrt{3}}\left[\boldsymbol{I}_{iz}\boldsymbol{I}_{jz} + \frac{1}{2}(\boldsymbol{I}_{i+}\boldsymbol{I}_{j-} + \boldsymbol{I}_{i-}\boldsymbol{I}_{j+})\right] = -\frac{1}{\sqrt{3}}[\boldsymbol{I}_i \bullet \boldsymbol{I}_j]} \qquad (10\text{-}20)$$

0.0.5.2 TENSOR PRODUCT GENERATION OF $T_{1,m}(i,j)$

The $l = 1$ components are, from equation equation (10-17) on page 216,

$$\boldsymbol{T}_{1,M}(i,j) = (-1)^{2+1-M}\sqrt{2(1)+1}\sum_{m1}^{+1}\sum_{m2}^{+1}\boldsymbol{T}1_{1,m1}(i)\boldsymbol{T}2_{1,m2}(j)\begin{pmatrix} 1 & 1 & 1 \\ M & -m1 & -m2 \end{pmatrix} \quad .$$

$$\boldsymbol{T}_{1,M}(i,j) = (-1)^{M+1}\sqrt{3}\sum_{m1}^{+1}\sum_{m2}^{+1}\boldsymbol{T}1_{1,m1}(i)\boldsymbol{T}2_{1,m2}(j)\begin{pmatrix} 1 & 1 & 1 \\ M & -m1 & -m2 \end{pmatrix} \qquad (10\text{-}21)$$

This formula applies to three components, M = 1, 0, and -1.

$$\boldsymbol{T}_{1,0}(i,j) = (-\sqrt{3})\left[\boldsymbol{T}1_{1,0}(i)\boldsymbol{T}2_{1,0}(j)\begin{pmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \end{pmatrix} + \boldsymbol{T}1_{1,1}(i)\boldsymbol{T}2_{1,-1}(j)\begin{pmatrix} 1 & 1 & 1 \\ 0 & -1 & 1 \end{pmatrix} + \boldsymbol{T}1_{1,-1}(i)\boldsymbol{T}2_{1,1}(j)\begin{pmatrix} 1 & 1 & 1 \\ 0 & 1 & -1 \end{pmatrix}\right]$$

$$\boldsymbol{T}_{1,1}(i,j) = \sqrt{3}\left[\boldsymbol{T}1_{1,0}(i)\boldsymbol{T}2_{1,1}(j)\begin{pmatrix} 1 & 1 & 1 \\ 1 & 0 & -1 \end{pmatrix} + \boldsymbol{T}1_{1,1}(i)\boldsymbol{T}2_{1,0}(j)\begin{pmatrix} 1 & 1 & 1 \\ 1 & -1 & 0 \end{pmatrix}\right]$$

$$\boldsymbol{T}_{1,-1}(i,j) = \sqrt{3}\left[\boldsymbol{T}1_{1,0}(i)\boldsymbol{T}2_{1,-1}(j)\begin{pmatrix} 1 & 1 & 1 \\ -1 & 0 & 1 \end{pmatrix} + \boldsymbol{T}1_{1,-1}(i)\boldsymbol{T}2_{1,0}(j)\begin{pmatrix} 1 & 1 & 1 \\ -1 & 1 & 0 \end{pmatrix}\right]$$

The Wigner 3-j coefficients are simply multiplied by the factor $(-1)^{a+b+c}$ upon the interchange of two adjacent rows or columns[1] in the corresponding symbol, so these three equations slightly simplify.

1. See Brink and Satchler, bottom of page 113 in APPENDIX I.

$$\boldsymbol{T}_{1,0}(i,j) = (-\sqrt{3})\left[\boldsymbol{T}1_{1,0}(i)\boldsymbol{T}2_{1,0}(j)\begin{pmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \end{pmatrix} + \begin{pmatrix} 1 & 1 & 1 \\ 0 & -1 & 1 \end{pmatrix}[\boldsymbol{T}1_{1,1}(i)\boldsymbol{T}2_{1,-1}(j) - \boldsymbol{T}1_{1,-1}(i)\boldsymbol{T}2_{1,1}(j)]\right]$$

$$\boldsymbol{T}_{1,1}(i,j) = \sqrt{3}\left[\begin{pmatrix} 1 & 1 & 1 \\ 0 & -1 & 1 \end{pmatrix}[\boldsymbol{T}1_{1,0}(i)\boldsymbol{T}2_{1,1}(j) - \boldsymbol{T}1_{1,1}(i)\boldsymbol{T}2_{1,0}(j)]\right]$$

$$\boldsymbol{T}^{(2)}_{1,-1}(i,j) = -\sqrt{3}\left[\begin{pmatrix} 1 & 1 & 1 \\ 0 & -1 & 1 \end{pmatrix}[\boldsymbol{T}1_{1,0}(i)\boldsymbol{T}2_{1,-1}(j) - \boldsymbol{T}1_{1,-1}(i)\boldsymbol{T}2_{1,0}(j)]\right]$$

The two coefficients needed for evaluation of the three components are computed as follows.

$$\begin{pmatrix} a & a & 1 \\ \alpha & -\alpha & 0 \end{pmatrix} = (-1)^{a-\alpha}\frac{\alpha}{\sqrt{a(a+1)(2a+1)}} \rightarrow \begin{pmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \end{pmatrix} = (-1)^{-1}\frac{0}{\sqrt{1(2)(3)}} = 0$$

$$\begin{pmatrix} 1 & a & a \\ 0 & -\alpha & \alpha \end{pmatrix} = (-1)^{a-\alpha+1}\frac{a}{\sqrt{a(a+1)(2a+1)}} \rightarrow \begin{pmatrix} 1 & 1 & 1 \\ 0 & -1 & 1 \end{pmatrix} = (-1)^{1}\frac{1}{\sqrt{1(2)(3)}} = \frac{-1}{\sqrt{6}}$$

Substitution into our three previous equations produces

$$\boldsymbol{T}_{1,0}(i,j) = (-\sqrt{3})\left[\boldsymbol{I}_{iz}\boldsymbol{I}_{jz}(0) + \frac{-1}{\sqrt{6}}\left[\left(-\frac{1}{\sqrt{2}}\boldsymbol{I}_{i+}\right)\left(\frac{1}{\sqrt{2}}\boldsymbol{I}_{j-}\right) - \left(\frac{1}{\sqrt{2}}\boldsymbol{I}_{i-}\right)\left(-\frac{1}{\sqrt{2}}\boldsymbol{I}_{j+}\right)\right]\right]$$

$$\boldsymbol{T}_{1,1}(i,j) = \sqrt{3}\left[\left(\frac{-1}{\sqrt{6}}\right)\left[\boldsymbol{I}_{iz}\left(-\frac{1}{\sqrt{2}}\boldsymbol{I}_{j+}\right) - \left(-\frac{1}{\sqrt{2}}\boldsymbol{I}_{i+}\right)\boldsymbol{I}_{jz}\right]\right] \qquad \boldsymbol{T}_{1,-1}(i,j) = -\sqrt{3}\left[\left(\frac{-1}{\sqrt{6}}\right)\left[\boldsymbol{I}_{iz}\left(\frac{1}{\sqrt{2}}\boldsymbol{I}_{j-}\right) - \left(\frac{1}{\sqrt{2}}\boldsymbol{I}_{i-}\right)\boldsymbol{I}_{jz}\right]\right]$$

and these slightly simplify into the formulas to be used throughout GAMMA.

$$\boldsymbol{T}_{1,0}(i,j) = \frac{-1}{2\sqrt{2}}(\boldsymbol{I}_{i+}\boldsymbol{I}_{j-} - \boldsymbol{I}_{i-}\boldsymbol{I}_{j+})$$

$$\boldsymbol{T}_{1,1}(i,j) = \frac{1}{2}[\boldsymbol{I}_{iz}\boldsymbol{I}_{j+} - \boldsymbol{I}_{i+}\boldsymbol{I}_{jz}] \qquad \boldsymbol{T}_{1,-1}(i,j) = \frac{1}{2}[\boldsymbol{I}_{iz}\boldsymbol{I}_{j-} - \boldsymbol{I}_{i-}\boldsymbol{I}_{jz}] \tag{18-2}$$

Note that an switch in the ordering of the terms for these elements will result in a sign change as these are commonly found in the literature. Also found in the literature is an alternate form which utilizes the Cartesian angular momentum components rather that the ladder operators.

$$\begin{aligned}\boldsymbol{T}_{1,0}(i,j) &= \frac{-1}{2\sqrt{2}}(\boldsymbol{I}_{i+}\boldsymbol{I}_{j-} - \boldsymbol{I}_{i-}\boldsymbol{I}_{j+}) = \frac{-1}{2\sqrt{2}}[(\boldsymbol{I}_{ix} + iI_{iy})(\boldsymbol{I}_{jx} - iI_{jy}) - (\boldsymbol{I}_{ix} - iI_{iy})(\boldsymbol{I}_{jx} + iI_{jy})] \\ &= \frac{-1}{2\sqrt{2}}[(\boldsymbol{I}_{ix}\boldsymbol{I}_{jx} - i\boldsymbol{I}_{ix}I_{jy} + iI_{iy}\boldsymbol{I}_{jx} + I_{iy}I_{jy}) - (\boldsymbol{I}_{ix}\boldsymbol{I}_{jx} + i\boldsymbol{I}_{ix}I_{jy} - iI_{iy}\boldsymbol{I}_{jx} + I_{iy}I_{jy})] \\ &= \frac{-1}{2\sqrt{2}}[(-i\boldsymbol{I}_{ix}I_{jy} + iI_{iy}\boldsymbol{I}_{jx}) - (i\boldsymbol{I}_{ix}I_{jy} - iI_{iy}\boldsymbol{I}_{jx})] = \frac{-1}{\sqrt{2}}[(-i\boldsymbol{I}_{ix}I_{jy} + iI_{iy}\boldsymbol{I}_{jx})] \\ &= \frac{i}{\sqrt{2}}[\boldsymbol{I}_{ix}I_{jy} - I_{iy}\boldsymbol{I}_{jx}]\end{aligned}$$

$$\begin{aligned}\boldsymbol{T}_{1,\pm1}(i,j) &= \frac{1}{2}[\boldsymbol{I}_{iz}\boldsymbol{I}_{j\pm} - \boldsymbol{I}_{i\pm}\boldsymbol{I}_{jz}] = \frac{1}{2}[\boldsymbol{I}_{iz}(\boldsymbol{I}_{jx} \pm iI_{jy}) - (\boldsymbol{I}_{ix} \pm iI_{iy})\boldsymbol{I}_{jz}] \\ &= \frac{1}{2}[(\boldsymbol{I}_{iz}\boldsymbol{I}_{jx} \pm i\boldsymbol{I}_{iz}I_{jy}) - (\boldsymbol{I}_{ix}\boldsymbol{I}_{jz} \pm iI_{iy}\boldsymbol{I}_{jz})] \\ &= \frac{1}{2}[(\boldsymbol{I}_{iz}\boldsymbol{I}_{jx} - \boldsymbol{I}_{ix}\boldsymbol{I}_{jz}) \pm i(i\boldsymbol{I}_{iz}I_{jy} - I_{iy}\boldsymbol{I}_{jz})]\end{aligned}$$

0.0.5.3 TENSOR PRODUCT GENERATION OF $T_{2,m}(i,j)$

Finally, the $l = 2$ components are, from equation (10-17),

$$\boldsymbol{T}_{2,M}(i,j) = (-1)^{2+2-M}\sqrt{2(2)+1}\sum_{m1}^{+1}\sum_{m2}^{+1}\boldsymbol{T}1_{1,m1}(i)\boldsymbol{T}2_{1,m2}(j)\begin{pmatrix} 2 & 1 & 1 \\ M & -m1 & -m2 \end{pmatrix} .$$

$$\boldsymbol{T}_{2,M}(i,j) = (-1)^{M}\sqrt{5}\sum_{m1}^{+1}\sum_{m2}^{+1}\boldsymbol{T}1_{1,m1}(i)\boldsymbol{T}2_{1,m2}(j)\begin{pmatrix} 2 & 1 & 1 \\ M & -m1 & -m2 \end{pmatrix} \qquad (10\text{-}22)$$

This formula applies to five components, M = 0, 1, -1, 2 and -2.

$$\boldsymbol{T}_{2,0}(i,j) = \sqrt{5}\left[\boldsymbol{T}1_{1,0}(i)\boldsymbol{T}2_{1,0}(j)\begin{pmatrix} 2 & 1 & 1 \\ 0 & 0 & 0 \end{pmatrix} + \boldsymbol{T}1_{1,1}(i)\boldsymbol{T}2_{1,-1}(j)\begin{pmatrix} 2 & 1 & 1 \\ 0 & -1 & 1 \end{pmatrix} + \boldsymbol{T}1_{1,-1}(i)\boldsymbol{T}2_{1,1}(j)\begin{pmatrix} 2 & 1 & 1 \\ 0 & 1 & -1 \end{pmatrix}\right]$$

$$\boldsymbol{T}_{2,1}(i,j) = -\sqrt{5}\left[\boldsymbol{T}1_{1,0}(i)\boldsymbol{T}2_{1,1}(j)\begin{pmatrix} 2 & 1 & 1 \\ 1 & 0 & -1 \end{pmatrix} + \boldsymbol{T}1_{1,1}(i)\boldsymbol{T}2_{1,0}(j)\begin{pmatrix} 2 & 1 & 1 \\ 1 & -1 & 0 \end{pmatrix}\right]$$

$$\boldsymbol{T}_{2,-1}(i,j) = -\sqrt{5}\left[\boldsymbol{T}1_{1,0}(i)\boldsymbol{T}2_{1,-1}(j)\begin{pmatrix} 2 & 1 & 1 \\ -1 & 0 & 1 \end{pmatrix} + \boldsymbol{T}1_{1,-1}(i)\boldsymbol{T}2_{1,0}(j)\begin{pmatrix} 2 & 1 & 1 \\ -1 & 1 & 0 \end{pmatrix}\right]$$

$$\boldsymbol{T}_{2,2}(i,j) = \sqrt{5}\left[\boldsymbol{T}1_{1,1}(i)\boldsymbol{T}2_{1,1}(j)\begin{pmatrix} 2 & 1 & 1 \\ 2 & -1 & -1 \end{pmatrix}\right] \qquad \boldsymbol{T}_{2,-2}(i,j) = \sqrt{5}\left[\boldsymbol{T}1_{1,-1}(i)\boldsymbol{T}2_{1,-1}(j)\begin{pmatrix} 2 & 1 & 1 \\ -2 & 1 & 1 \end{pmatrix}\right]$$

Again using the fact that an interchange of two adjacent columns multiplies the coefficient by the factor $(-1)^{a+b+c}$ as well as the identity

$$\begin{pmatrix} a & b & c \\ -\alpha & -\beta & -\gamma \end{pmatrix} = (-1)^{a+b+c}\begin{pmatrix} a & b & c \\ \alpha & \beta & \gamma \end{pmatrix} , \qquad (10\text{-}23)$$

we obtain the five equations

$$\boldsymbol{T}_{2,0}(i,j) = \sqrt{5}\left[\begin{pmatrix} 2 & 1 & 1 \\ 0 & 0 & 0 \end{pmatrix}\boldsymbol{T}1_{1,0}(i)\boldsymbol{T}2_{1,0}(j) + \begin{pmatrix} 2 & 1 & 1 \\ 0 & -1 & 1 \end{pmatrix}[\boldsymbol{T}1_{1,1}(i)\boldsymbol{T}2_{1,-1}(j) + \boldsymbol{T}1_{1,-1}(i)\boldsymbol{T}2_{1,1}(j)]\right]$$

$$\boldsymbol{T}_{2,1}(i,j) = -\sqrt{5}\left[\begin{pmatrix} 2 & 1 & 1 \\ 1 & 0 & -1 \end{pmatrix}[\boldsymbol{T}1_{1,0}(i)\boldsymbol{T}2_{1,1}(j) + \boldsymbol{T}1_{1,1}(i)\boldsymbol{T}2_{1,0}(j)]\right]$$

$$\boldsymbol{T}_{2,-1}(i,j) = -\sqrt{5}\left[\begin{pmatrix} 2 & 1 & 1 \\ -1 & 0 & 1 \end{pmatrix}[\boldsymbol{T}1_{1,0}(i)\boldsymbol{T}2_{1,-1}(j) + \boldsymbol{T}1_{1,-1}(i)\boldsymbol{T}2_{1,0}(j)]\right]$$

$$\boldsymbol{T}_{2,2}(i,j) = \sqrt{5}\left[\boldsymbol{T}1_{1,1}(i)\boldsymbol{T}2_{1,1}(j)\begin{pmatrix} 2 & 1 & 1 \\ 2 & -1 & -1 \end{pmatrix}\right] \qquad \boldsymbol{T}_{2,-2}(i,j) = \sqrt{5}\left[\boldsymbol{T}1_{1,-1}(i)\boldsymbol{T}2_{1,-1}(j)\begin{pmatrix} 2 & 1 & 1 \\ 2 & -1 & -1 \end{pmatrix}\right]$$

Inserting the rank 1 tensor components in equation (10-18), we obtain

$$T_{2,0}(i,j) = (\sqrt{5})\left[\begin{pmatrix} 2 & 1 & 1 \\ 0 & 0 & 0 \end{pmatrix} I_{iz}I_{jz} + \left(-\frac{1}{2}\right)\begin{pmatrix} 2 & 1 & 1 \\ 0 & -1 & 1 \end{pmatrix}[I_{i+}I_{j-} + I_{i-}I_{j+}]\right]$$

$$T_{2,1}(i,j) = \sqrt{\frac{5}{2}}\begin{pmatrix} 2 & 1 & 1 \\ 1 & 0 & -1 \end{pmatrix}[I_{iz}I_{j+} + I_{i+}I_{jz}] \qquad T_{2,-1}(i,j) = -\sqrt{\frac{5}{2}}\begin{pmatrix} 2 & 1 & 1 \\ 1 & 0 & -1 \end{pmatrix}[I_{iz}I_{j-} + I_{i-}I_{jz}]$$

$$T_{2,2}(i,j) = \sqrt{\frac{5}{4}}\left[I_{i+}I_{j+}\begin{pmatrix} 2 & 1 & 1 \\ 2 & -1 & -1 \end{pmatrix}\right] \qquad T_{2,-2}(i,j) = \sqrt{\frac{5}{4}}\left[I_{i-}I_{j-}\begin{pmatrix} 2 & 1 & 1 \\ 2 & -1 & -1 \end{pmatrix}\right]$$

The four Wigner coefficients which occur in the previous set of formulae are obtained as follows

$$\begin{pmatrix} a & a & 2 \\ \alpha & -\alpha & 0 \end{pmatrix} = (-1)^{a-\alpha}\frac{3\alpha^2 - a(a+1)}{\sqrt{a(a+1)(2a+3)(2a+1)(2a-1)}} \rightarrow \begin{pmatrix} 1 & 1 & 2 \\ 0 & 0 & 0 \end{pmatrix} = \begin{pmatrix} 2 & 1 & 1 \\ 0 & 0 & 0 \end{pmatrix} = \frac{2}{\sqrt{30}}$$

$$\begin{pmatrix} a & a & 2 \\ \alpha & -\alpha & 0 \end{pmatrix} = (-1)^{a-\alpha}\frac{3\alpha^2 - a(a+1)}{\sqrt{a(a+1)(2a+3)(2a+1)(2a-1)}} \rightarrow \begin{pmatrix} 1 & 1 & 2 \\ 1 & -1 & 0 \end{pmatrix} = \begin{pmatrix} 2 & 1 & 1 \\ 0 & -1 & 1 \end{pmatrix} = \frac{1}{\sqrt{30}}$$

$$\begin{pmatrix} a & a & 2 \\ \alpha & -\alpha - 1 & 1 \end{pmatrix} = (-1)^{a-\alpha}(2\alpha+1)\sqrt{\frac{3(a-\alpha)(a+\alpha+1)}{a(2a+3)(2a+2)(2a+1)(2a-1)}} \rightarrow \begin{pmatrix} 1 & 1 & 2 \\ 0 & -1 & 2 \end{pmatrix} = \begin{pmatrix} 2 & 1 & 1 \\ 1 & 0 & -1 \end{pmatrix} = -\sqrt{\frac{1}{10}}$$

$$\begin{pmatrix} a & a & 2 \\ 1 & 1 & -2 \end{pmatrix} = \begin{pmatrix} a & a & 2 \\ 0 & 0 & 0 \end{pmatrix}\sqrt{\frac{6}{4}} \rightarrow \begin{pmatrix} 2 & 1 & 1 \\ 2 & -1 & -1 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 2 \\ -1 & -1 & 2 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 2 \\ 1 & 1 & -2 \end{pmatrix} = \sqrt{\frac{6}{4}}\begin{pmatrix} 1 & 1 & 2 \\ 0 & 0 & 0 \end{pmatrix} = \frac{1}{\sqrt{5}}$$

Substituting these coefficients into the previous equations produces the following.

$$T_{2,0}(i,j) = \frac{1}{\sqrt{6}}\left[2I_{iz}I_{jz} - \frac{1}{2}(I_{i+}I_{j-} + I_{i-}I_{j+})\right] = \frac{1}{\sqrt{6}}\left[3I_{iz}I_{jz} - I_{iz}I_{jz} - \frac{1}{2}(I_{i+}I_{j-} + I_{i-}I_{j+})\right]$$

$$T_{2,1}(i,j) = \sqrt{\frac{5}{2}}\left(-\sqrt{\frac{1}{10}}\right)[I_{iz}I_{j+} + I_{i+}I_{jz}] \qquad T_{2,-1}(i,j) = -\sqrt{\frac{5}{2}}\left(-\sqrt{\frac{1}{10}}\right)[I_{iz}I_{j-} + I_{i-}I_{jz}]$$

$$T_{2,2}(i,j) = \sqrt{\frac{5}{4}}\left[I_{i+}I_{j+}\left(\frac{1}{\sqrt{5}}\right)\right] \qquad T_{2,-2}(i,j) = \sqrt{\frac{5}{4}}\left[I_{i-}I_{j-}\left(\frac{1}{\sqrt{5}}\right)\right]$$

Thus we have produced five components, and these are in agreement with those derived in the previous section using ladder operators (See equation on page page 210.).

$$T_{2,0}(i,j) = \frac{1}{\sqrt{6}}[3I_{iz}I_{jz} - (I_i \bullet I_j)]$$

$$T_{2,\pm1}(i,j) = \mp\frac{1}{2}[I_{iz}I_{j\pm} + I_{i\pm}I_{jz}] \qquad T_{2,\pm2}(i,j) = \frac{1}{2}I_{i\pm}I_{j\pm} \qquad (18\text{-}3)$$

10.11.8Summary of GAMMA Rank 2 Tensor Components

For convenience we regroup the rank 2 spin tensor components into one location. This are taken from equations equation (10-20) on page 217, equation (18-2) on page 218, and (18-3)

Rank 2 Irreducible Spherical Spin Tensor Components

$$T_{0,0}(ij) = \frac{-1}{\sqrt{3}}\left[I_{iz}I_{jz} + \frac{1}{2}(I_{i+}I_{j-} + I_{i-}I_{j+})\right]$$

$$T_{1,0}(ij) = \frac{-1}{2\sqrt{2}}[I_{i+}I_{j-} - I_{i-}I_{j+}] \qquad T_{1,\pm1}(ij) = \frac{-1}{2}[I_{i\pm}\ I_{jz} - I_{iz}I_{j\pm}\ ]$$

$$T_{2,0}(ij) = \frac{1}{\sqrt{6}}[3I_{iz}I_{jz} - (\boldsymbol{I}_i \bullet \boldsymbol{I}_j)]$$

$$T_{2,\pm1}(ij) = \mp\frac{1}{2}[I_{i\pm}\ I_{jz} + I_{iz}I_{j\pm}\ ] \qquad T_{2,\pm2}(ij) = \frac{1}{2}[I_{i\pm}\ I_{j\pm}\ ]$$

Figure E General formulae for the rank 2 irreducible spherical spin tensor components.

When treating spin angular momentum where both spins have I=1/2 the matrix representations of these tensor components (operators) are given in the following figure. They are shown in the composite Hilbert space of the two spins (spin indices are implicit here)[1].

Rank 2 Irreducible Spherical Spin Tensor Components - Matrix Representations

$$T^{(2)}_{0,0} = \frac{-1}{4\sqrt{3}}\begin{bmatrix}1&0&0&0\\0&-1&2&0\\0&2&-1&0\\0&0&0&1\end{bmatrix} \qquad T^{(2)}_{1,0} = \frac{-1}{2\sqrt{2}}\begin{bmatrix}0&0&0&0\\0&0&1&0\\0&-1&0&0\\0&0&0&0\end{bmatrix} \qquad T^{(2)}_{1,-1} = \frac{1}{4}\begin{bmatrix}0&0&0&0\\1&0&0&0\\-1&0&0&0\\0&1&-1&0\end{bmatrix} \qquad T^{(2)}_{1,1} = \frac{1}{4}\begin{bmatrix}0&1&-1&0\\0&0&0&1\\0&0&0&-1\\0&0&0&0\end{bmatrix}$$

$$T^{(2)}_{2,0} = \frac{1}{2\sqrt{6}}\begin{bmatrix}1&0&0&0\\0&-1&-1&0\\0&-1&-1&0\\0&0&0&1\end{bmatrix} \quad T^{(2)}_{2,1} = \frac{1}{4}\begin{bmatrix}0&-1&-1&0\\0&0&0&1\\0&0&0&1\\0&0&0&0\end{bmatrix} \quad T^{(2)}_{2,-1} = \frac{1}{4}\begin{bmatrix}0&0&0&0\\1&0&0&0\\1&0&0&0\\0&-1&-1&0\end{bmatrix} \quad T^{(2)}_{2,-2} = \frac{1}{2}\begin{bmatrix}0&0&0&0\\0&0&0&0\\0&0&0&0\\1&0&0&0\end{bmatrix} \quad T^{(2)}_{2,2} = \frac{1}{2}\begin{bmatrix}0&0&0&1\\0&0&0&0\\0&0&0&0\\0&0&0&0\end{bmatrix}$$

Figure F Matrix representations in a 2-spin Hilbert space (both I=1/2) of the rank 2 irreducible spherical spin tensor components.

Having shown the form of the two spin rank 2 tensor components we now show the case when the rank 2 “spin” tensor actually contains both spatial and spin terms[2]. Here the tensor will involve

1. The GAMMA program which produced these tensor components is listed at the end of this Chapter entitled Rank2_SpinT.cc. Alternatively, GAMMA can generate these components via tensor products of two rank 1 tensors. This latter computation may be performed with the program XProd_SpinT.cc also listed.

only one spin and its components are specified as $T_{l,m}(i)$. The nine formulas which produce these components are given by[1]

Rank 2 Irreducible Spherical Spin-Space Tensor Components

$$T_{0,0}(ij) = \frac{-1}{\sqrt{3}}\left[I_{iz}u_z + \frac{1}{2}(I_{i+}u_- + I_{i-}u_+)\right] = \frac{-1}{\sqrt{3}}\grave{I}_i \bullet \grave{u}$$

$$T_{1,0}(ij) = \frac{-1}{2\sqrt{2}}[I_{i+}u_- - I_{i-}u_+] \qquad T_{1,\pm1}(ij) = \frac{-1}{2}[I_{i\pm}\,u_z - I_{iz}u_\pm\,]$$

$$T_{2,0}(ij) = \frac{1}{\sqrt{6}}[3I_{iz}u_z - (\grave{I}_i \bullet \grave{u})]$$

$$T_{2,\pm1}(ij) = \mp\frac{1}{2}[I_{i\pm}\,u_z + I_{iz}u_\pm\,] \qquad T_{2,\pm2}(ij) = \frac{1}{2}[I_{i\pm}\,u_\pm\,]$$

Figure G General formulae for the rank 2 irreducible spherical spin tensor components.

where $\grave{u}$ is a normalized vector $\grave{u} = u_x\mathbf{i} + u_y\mathbf{j} + u_z\mathbf{k}$. We have replaced the terms in arising from the second momentum (vector) operator $\grave{I}_j$ with the vector $\grave{u}$. The simplest situation occurs when $\grave{u}$ points along the positive z-axis, $\grave{u} = \grave{k}$. The applicable equations are under this constraint are listed in the following figure.

Aligned Rank 2 Irreducible Spherical Spin-Space Tensor Components

$$T_{0,0}(ij) = \frac{-1}{\sqrt{3}}I_{iz}$$

$$T_{1,0}(ij) = 0 \qquad T_{1,\pm1}(ij) = \frac{-1}{2}I_{i\pm}$$

$$T_{2,0}(ij) = \frac{2}{\sqrt{6}}I_{iz}$$

$$T_{2,\pm1}(ij) = \mp\frac{1}{2}I_{i\pm} \qquad T_{2,\pm2}(ij) = 0$$

Figure H Formulae for the rank 2 irreducible spherical space-spin tensor components when the spatial vector is aligned along the +z axis.

For a spin 1/2 particle and $\grave{u}$ along the positive z-axis the matrix form of these tensor components are shown in the following figure[2] (in the single spin Hilbert space).

2. Used to treat liquid relaxation by chemical shift anisotropy (CSA) for example.

1. For these formulae, it is important to note that it is the second component which is set to the vector $\grave{u}$, although we might just as well have used the first vector instead. The difference is that the $l = 1$ equations would then appear of opposite sign from those given here.

Aligned Rank 2 Irreducible Spherical Spin-Space Tensor - Matrix Representations

Rank 2 Irreducible Spherical Spin-Space Tensor Components Matrix Representations in 1-spin (I=1/2) Hilbert Space

$$T^{(2)}_{0,0} = \frac{-1}{2\sqrt{3}}\begin{bmatrix}1 & 0\\ 0 & -1\end{bmatrix} \qquad T^{(2)}_{1,0} = \begin{bmatrix}0 & 0\\ 0 & 0\end{bmatrix} \qquad T^{(2)}_{1,-1} = \frac{-1}{2}\begin{bmatrix}0 & 0\\ 1 & 0\end{bmatrix} \qquad T^{(2)}_{1,1} = \frac{-1}{2}\begin{bmatrix}0 & 1\\ 0 & 0\end{bmatrix}$$

$$T^{(2)}_{2,0} = \frac{1}{\sqrt{6}}\begin{bmatrix}1 & 0\\ 0 & -1\end{bmatrix} \quad T^{(2)}_{2,1} = \frac{-1}{2}\begin{bmatrix}0 & 1\\ 0 & 0\end{bmatrix} \quad T^{(2)}_{2,-1} = \frac{1}{2}\begin{bmatrix}0 & 0\\ 1 & 0\end{bmatrix} \quad T^{(2)}_{2,-2} = \begin{bmatrix}0 & 0\\ 0 & 0\end{bmatrix} \quad T^{(2)}_{2,2} = \begin{bmatrix}0 & 0\\ 0 & 0\end{bmatrix}$$

Figure I Matrix representations in a single spin (I=1/2) Hilbert space of the rank 2 irreducible spherical spin-space tensor components when the spatial vector is aligned along the +z axis.

One must be very careful in using single spin rank 2 tensors of this type because they contain both spatial and spin components. It is improper to rotate this tensor in spin space because it also rotates spatial variables. If some Hamiltonian is a product of a spatial tensor and this tensor, the spatial tensor cannot be rotated as it rotates only part of the spatial components[1]. Furthermore, it must be noted that **these are not the same as the single spin rank 1 tensor components**.

At this point we briefly return to our original formulation of a general Hamiltonian. It was stated that we needed to express our Cartesian tensor in terms of their irreducible spherical components which behave in a well defined manner under a rotation. The above set of equations does just that! We can go one step further and show explicitly how the components of the Cartesian spin tensors in equation (10-6) can be used to directly produce the components of the irreducible spherical spin tensors. We have

$$\begin{bmatrix}I_{jx}\\ I_{jy}\\ I_{jz}\end{bmatrix} \bullet \begin{bmatrix}I_{ix} & I_{iy} & I_{iz}\end{bmatrix} = \begin{bmatrix}I_{xj}I_{xi} & I_{xj}I_{yi} & I_{xj}I_{zi}\\ I_{yj}I_{xi} & I_{yj}I_{yi} & I_{yj}I_{zi}\\ I_{zj}I_{xi} & I_{zj}I_{yi} & I_{zj}I_{zi}\end{bmatrix} \quad \text{and} \quad \begin{bmatrix}B_x\\ B_y\\ B_z\end{bmatrix} \bullet \begin{bmatrix}I_{ix} & I_{iy} & I_{iz}\end{bmatrix} = \begin{bmatrix}B_xI_{xi} & B_xI_{yi} & B_xI_{zi}\\ B_yI_{xi} & B_yI_{yi} & B_yI_{zi}\\ B_zI_{xi} & B_zI_{yi} & B_zI_{zi}\end{bmatrix}$$

Using a more general form

$$\begin{bmatrix}I_x\\ I_y\\ I_z\end{bmatrix} \bullet \begin{bmatrix}S_x & S_y & S_z\end{bmatrix} = \begin{bmatrix}I_xS_x & I_xS_y & I_xS_z\\ I_yS_x & I_yS_y & I_yS_z\\ I_zS_x & I_zS_y & I_zS_z\end{bmatrix} = \hat{X}$$

the following relationships hold true[2]

2. The GAMMA program which produced these matrix representations can be found at the end of this Chapter, Rank2SS_SpinT.cc.
1. See the discussion in Mehring

$$\boldsymbol{X}_{0,0} = -\frac{1}{\sqrt{3}}[\boldsymbol{I}_x\boldsymbol{S}_x + \boldsymbol{I}_y\boldsymbol{S}_y + \boldsymbol{I}_z\boldsymbol{S}_z] = -\frac{1}{\sqrt{3}}[\boldsymbol{I} \bullet \boldsymbol{S}]$$

$$\boldsymbol{X}_{1,0} = \frac{i}{\sqrt{2}}[\boldsymbol{I}_z\boldsymbol{S}_x - \boldsymbol{I}_y\boldsymbol{S}_x] \qquad \boldsymbol{X}_{1,\pm1} = \frac{1}{2}[(\boldsymbol{I}_x\boldsymbol{S}_y - \boldsymbol{I}_x\boldsymbol{S}_z) \pm i(\boldsymbol{I}_z\boldsymbol{S}_y - \boldsymbol{I}_y\boldsymbol{S}_z)]$$

$$\boldsymbol{X}_{2,0} = \frac{1}{\sqrt{6}}[3\boldsymbol{I}_z\boldsymbol{S}_z - (\boldsymbol{I}_x\boldsymbol{S}_x + \boldsymbol{I}_y\boldsymbol{S}_y + \boldsymbol{I}_z\boldsymbol{S}_z)] = \frac{1}{\sqrt{6}}[3\boldsymbol{I}_z\boldsymbol{S}_z - \boldsymbol{I} \bullet \boldsymbol{S}]$$

$$\boldsymbol{X}_{2,\pm1} = \mp\frac{1}{2}[(\boldsymbol{I}_x\boldsymbol{S}_z + \boldsymbol{I}_z\boldsymbol{S}_x) \pm i(\boldsymbol{I}_y\boldsymbol{S}_z + \boldsymbol{I}_z\boldsymbol{S}_y)] = \mp\frac{1}{2}[\boldsymbol{I}_z\boldsymbol{S}_\pm + \boldsymbol{I}_\pm\boldsymbol{S}_z]$$

$$\boldsymbol{X}_{2,\pm2} = -\frac{1}{\sqrt{3}}[(\boldsymbol{I}_x\boldsymbol{S}_x - \boldsymbol{I}_y\boldsymbol{S}_y) \pm i(\boldsymbol{I}_x\boldsymbol{S}_y + \boldsymbol{I}_y\boldsymbol{S}_x)] = \frac{1}{2}\boldsymbol{I}_\pm\boldsymbol{S}_\pm$$

2. An alternative form for $\boldsymbol{X}_{2,0}$ is $\boldsymbol{X}_{2,0} = \frac{1}{2\sqrt{6}}[4\boldsymbol{I}_z\boldsymbol{S}_z - (\boldsymbol{I}_+\boldsymbol{S}_- + \boldsymbol{I}_-\boldsymbol{S}_+)]$.

10.11.9Tensor Rotations

One way to define an irreducible spherical tensor is by its properties under rotations. A spherical irreducible tensor operator of rank ***l*** has ***2l+1*** components, $\boldsymbol{T_{l,m}}$, which transform under the ***2l+1*** dimensional representation of the rotation group ***R*** according to

$$\boldsymbol{R}\boldsymbol{T}_{l,m}\boldsymbol{R}^{-1} = \sum_{m'} D^{l}_{mm'}\boldsymbol{T}_{l,m'} \tag{10-24}$$

When a rotation of the coordinate system is performed (by ***R***) each $\boldsymbol{T_{l,m}}$ is transformed into a linear combination of the initial set of ***2l+1*** components, $\boldsymbol{T_{l,m'}}$.

$$T_{l,m} = RT_{l,m}R^{\dagger} = \sum_{m'} T_{l,m'}D_{m',m}(\alpha,\beta,\gamma) \tag{10-25}$$

In this equation, the $\boldsymbol{D_{m',m}(\alpha,\beta,\gamma)}$ are elements of the rank ***l*** Wigner rotation matrices and the set of angles $\{\alpha,\beta,\gamma\}$ are Euler angles for the rotation.

We shall now write down the corresponding results for rotations of the rank 1 and rank 2 spin tensor components. As we have already noted, the rank 2 spin-space tensors should not be rotated; they are fixed into the coordinate system of the spatial vector used in their construction. For rotation of a rank 1 spin tensor

$$\underset{\sim}{T}^{m}_{1} = \sum_{m'} T^{m'}_{1}D^{1}_{m',m}(\Omega_E) = T^{-1}_{1}D^{1}_{-1,m}(\Omega_E) + T^{0}_{1}D^{1}_{0,m}(\Omega_E) + T^{1}_{1}D^{1}_{1,m}(\Omega_E) \tag{10-26}$$

The Wigner rotation matrix elements are often expressed relative to reduced Wigner rotation matrix elements *via*

$$\boldsymbol{D}^{(1)}_{m,n}(\alpha,\beta,\gamma) = e^{-i\alpha m}d^{(1)}_{m,n}(\beta)e^{-i\gamma n} \; . \tag{10-27}$$

where the reduced (rank 1) Wigner rotation matrix elements are given in the following figure[1].

Reduced Rank 1 Wigner Rotation Matrix Elements $d^{(1)}_{m,n}(\beta)$

$d^{(1)}_{m,n}(\beta)$	$n = 1$	$n = 0$	$n = -1$
$m = 1$	$\cos^2(\beta/2)$	$-\sqrt{1/2}\sin\beta$	$\sin^2(\beta/2)$
$m = 0$	$\sqrt{1/2}\sin\beta$	$\cos\beta$	$-\sqrt{1/2}\sin\beta$
$m = -1$	$\sin^2(\beta/2)$	$\sqrt{1/2}\sin\beta$	$\cos^2(\beta/2)$

Figure J The reduced Wigner rotation matrix elements of rank 1. The angle β is the standard Eu-

1. See Brink and Satchler, page 24, TABLE 1. These are also detailed in the GAMMA User Documentation, see the Spatial Functions Chapter.

ler angle.

$$\begin{aligned}
\underset{\sim}{T}{}^m_1 &= \sum_{m'}^{\pm 1} T_1^{m'} D^1_{m',m}(\Omega_E) = T_1^{-1} D^1_{-1,m}(\Omega_E) + T_1^0 D^1_{0,m}(\Omega_E) + T_1^1 D^1_{1,m}(\Omega_E) \\
&= [T_1^{-1} e^{i\alpha} d^{(1)}_{-1,m}(\beta) + T_1^0 d^{(1)}_{0,m}(\beta) + T_1^1 e^{-i\alpha} d^{(1)}_{1,m}(\beta)] e^{-i\gamma m}
\end{aligned} \tag{10-28}$$

$$\begin{aligned}
\underset{\sim}{T}{}^1_1 &= [T_1^{-1} e^{i\alpha} d^{(1)}_{-1,1}(\beta) + T_1^0 d^{(1)}_{0,1}(\beta) + T_1^1 e^{-i\alpha} d^{(1)}_{1,1}(\beta)] e^{-i\gamma} \\
\underset{\sim}{T}{}^0_1 &= T_1^{-1} e^{i\alpha} d^{(1)}_{-1,0}(\beta) + T_1^0 d^{(1)}_{0,0}(\beta) + T_1^1 e^{-i\alpha} d^{(1)}_{1,0}(\beta) \\
\underset{\sim}{T}{}^{-1}_1 &= [T_1^{-1} e^{i\alpha} d^{(1)}_{-1,-1}(\beta) + T_1^0 d^{(1)}_{0,-1}(\beta) + T_1^1 e^{-i\alpha} d^{(1)}_{1,-1}(\beta)] e^{i\gamma}
\end{aligned}$$

$$\begin{aligned}
\underset{\sim}{T}{}^1_1 &= [T_1^{-1} e^{i\alpha} \sin^2(\beta/2) + \sqrt{1/2}\, T_1^0 \sin\beta + T_1^1 \cos^2(\beta/2)] e^{-i\gamma} \\
\underset{\sim}{T}{}^0_1 &= \sqrt{1/2}\, T_1^{-1} e^{i\alpha} \sin\beta + T_1^0 \cos\beta - \sqrt{1/2}\, T_1^1 e^{-i\alpha} \sin\beta \\
\underset{\sim}{T}{}^{-1}_1 &= [T_1^{-1} e^{i\alpha} \cos^2(\beta/2) - \sqrt{1/2}\, T_1^0 \sin\beta + T_1^1 e^{-i\alpha} \sin^2(\beta/2)] e^{i\gamma}
\end{aligned} \tag{10-29}$$

For rotation of a rank 2 spin tensor

$$\begin{aligned}
\underset{\sim}{T}{}^m_2 &= \sum_{m'}^{\pm 2} T_2^{m'} D^2_{m',m}(\Omega_E) \\
&= T_2^{-2} D^2_{-2,m}(\Omega_E) + T_2^{-1} D^2_{-1,m}(\Omega_E) + T_2^0 D^2_{0,m}(\Omega_E) + T_2^1 D^2_{1,m}(\Omega_E) + T_2^2 D^2_{2,m}(\Omega_E) \\
&= [T_2^{-2} e^{i2\alpha} d^2_{-2,m}(\beta) + T_2^{-1} e^{i\alpha} d^2_{-1,m}(\beta) \\
&\qquad + T_2^0 d^2_{0,m}(\beta) + T_2^1 e^{-i\alpha} d^2_{1,m}(\beta) + T_2^2 e^{-i2\alpha} d^2_{2,m}(\beta)] e^{-i\gamma m}
\end{aligned} \tag{10-30}$$

and the reduced rank 2 reduced Wigner rotation matrix elements are given in the following figure[1].

Reduced Rank 2 Wigner Rotation Matrix Elements $d^{(2)}_{m,n}(\beta)$

m	$n = 2$	$n = 1$	$n = 0$	$n = -1$	$n = -2$
2	$\cos^4(\beta/2)$	$-\frac{1}{2}(1+\cos\beta)\sin\beta$	$\sqrt{3/8}\sin^2\beta$	$\frac{1}{2}\sin\beta(\cos\beta-1)$	$\sin^4(\beta/2)$
1	$\frac{1}{2}\sin\beta(\cos\beta+1)$	$\cos^2\beta-\frac{1}{2}(1-\cos\beta)$	$-\sqrt{3/2}\sin\beta\cos\beta$	$\frac{1}{2}(1+\cos\beta)-\cos^2\beta$	$\frac{1}{2}\sin\beta(\cos\beta-1)$
0	$\sqrt{3/8}\sin^2\beta$	$\sqrt{3/8}\sin(2\beta)$	$\frac{1}{2}(3\cos^2\beta-1)$	$-\sqrt{3/8}\sin(2\beta)$	$\sqrt{3/8}\sin^2\beta$
−1	$-\frac{1}{2}(\cos\beta-1)\sin\beta$	$\frac{1}{2}(1+\cos\beta)-\cos^2\beta$	$\sqrt{3/2}\sin\beta\cos\beta$	$\cos^2\beta-\frac{1}{2}(1-\cos\beta)$	$-\frac{1}{2}(1+\cos\beta)\sin\beta$
−2	$\sin^4(\beta/2)$	$-\frac{1}{2}(\cos\beta-1)\sin\beta$	$\sqrt{3/8}\sin^2\beta$	$\frac{1}{2}(1+\cos\beta)\sin\beta$	$\cos^4(\beta/2)$

Figure K The reduced Wigner rotation matrix elements of rank 2. The angle β is the standard Euler angle.

1. See Brink and Satchler, page 24, TABLE 1.

10.11.10Clebsch-Gordan & Wigner-3j Coefficients

The Clebsch-Gordan coefficients are computed via the general formula[1]

$$\langle ab\alpha\beta|c\gamma\rangle = \delta_{\alpha+\beta,\gamma}\times\Delta(abc)\times[(2c+1)(a+\alpha)!(a-\alpha)!(b+\beta)!(b-\beta)!(c+\gamma)!(c-\gamma)!]^{1/2}$$
$$\times\sum_{\nu}(-1)^{\nu}[(a-\alpha-\nu)!(c-b+\alpha+\nu)!(b+\beta-\nu)!(c-a-\beta+\nu)!\nu!(a+b-c-\gamma)!]^{-1} \quad (10\text{-}31)$$

where the function $\Delta(abc)$ is obtained from

$$\Delta(abc) = \left[\frac{(a-b-c)!(a+c-b)!(b+c-a)!}{(a+b+c+1)!}\right]^{1/2} \quad (10\text{-}32)$$

and the summation runs from ***v*** equal to zero through the positive integers as long as all factorials are non-negative.

The Wigner 3-j coefficients are computed from the Clebsch-Gordan coefficients via the formula[2]

$$\begin{pmatrix} a & b & c \\ \alpha & \beta & \gamma \end{pmatrix} = (-1)^{\gamma+b-a}\frac{1}{\sqrt{2c+1}}\langle ab\alpha\beta|c\text{-}\gamma\rangle \quad (10\text{-}33)$$

Some of the Wigner 3-j coefficients are tabulated here for use as a check of this function. When $\alpha = \beta = \gamma = 0$ the equation that applies is[3]

$$\begin{pmatrix} a & b & c \\ 0 & 0 & 0 \end{pmatrix} = (-1)^{g}\Delta(abc)g![(g-a)!(g-b)!(g-c)!]^{-1} \quad (10\text{-}34)$$

where $2g = a+b+c$ and

$$\Delta(abc) = \left[\frac{(a-b-c)!(a+c-b)!(b+c-a)!}{(a+b+c+1)!}\right]^{1/2} .$$

From here the following table can be generated

1. A general formula for the Clebsch-Gordan coefficients is given in Brink and Satchler, page 34, equation (2.34). The relationship between the Clebsch-Gordan coefficients and Wigner 3-j coefficients is found on page 113 of their Appendix 1.
2. Brink and Satchler, page 39, equation (3.3). This formula is also found in their Appendix I, page 113.
3. Brink and Satchler, page 34, equation (2.35) combined with the relationship between the Clebsch-Gordan coefficients and the Wigner 3-j symbols.

Values of Wigner 3-j Coefficients $\begin{pmatrix} a & b & c \\ 0 & 0 & 0 \end{pmatrix}$

a b c	$\begin{pmatrix} a & b & c \\ 0 & 0 & 0 \end{pmatrix}$		a b c	$\begin{pmatrix} a & b & c \\ 0 & 0 & 0 \end{pmatrix}$		a b c	$\begin{pmatrix} a & b & c \\ 0 & 0 & 0 \end{pmatrix}$	
0 1 1	$-\sqrt{1/3}$	-0.577	1 5 6	$\sqrt{6/143}$	+0.205	3 3 6	$\sqrt{100/3003}$	+0.182
0 2 2	$\sqrt{1/5}$	+0.447	2 2 2	$-\sqrt{2/35}$	-0.239	3 4 5	$\sqrt{20/1001}$	+0.141
0 3 3	$-\sqrt{1/7}$	-0.378	2 2 4	$\sqrt{2/35}$	+0.239	3 5 6	$-\sqrt{7/429}$	-0.128
0 4 4	$\sqrt{1/9}$	+0.333	2 3 3	$\sqrt{4/105}$	+0.195	4 4 4	$\sqrt{18/1001}$	+0.134
0 5 5	$-\sqrt{1/11}$	-0.301	2 3 5	$-\sqrt{10/231}$	-0.208	4 4 6	$\sqrt{-\sqrt{20/1287}}$	-0.125
0 6 6	$\sqrt{1/13}$	+0.277	2 4 4	$-\sqrt{20/693}$	-0.170	4 5 6	$-\sqrt{2/143}$	-0118
1 1 2	$\sqrt{2/15}$	+0.365	2 4 6	$\sqrt{4/143}$	+0.187	4 6 6	$\sqrt{28/2431}$	+0.107
1 2 3	$-\sqrt{3/35}$	-0.293	2 5 5	$\sqrt{10/429}$	+0.153	5 5 6	$\sqrt{80/7293}$	+0.105
1 3 4	$\sqrt{4/63}$	+0.252	2 6 6	$-\sqrt{14/715}$	-0.140	6 6 6	$-\sqrt{400/46189}$	-0.093
1 4 5	$-\sqrt{5/99}$	-0.225	2 3 4	$-\sqrt{2/77}$	-0.161			

Figure L Values[1] of selected Wigner 3-j coefficients with $\alpha = \beta = \gamma = 0$.

Equation (10-33) can be very tedious formula to work with. Thus it is very helpful to have simple formulae for specific cases, such as (10-34), to facilitate the evaluation of these 3-j coefficients. Many such formulae are found in the literature[2] and a few which are particularly helpful are reproduced below.

1. Brink and Satchler, page 35, TABLE 2. Note that there is a sign error in their table on the 4 5 5 element as can be demonstrated from equation (10-34).
2. Brink and Satchler, page 36, TABLE 3.More discussion concerning these coefficients can be found in their Appendix 1 beginning on page 112.

Specific Wigner 3j Coefficient Equations

$$\begin{pmatrix} a & a+1 & 1 \\ \alpha & -\alpha & 0 \end{pmatrix} = (-1)^{a-\alpha-1}\sqrt{\frac{(a-\alpha+1)(a+\alpha+1)}{(a+1)(2a+1)(2a+3)}}$$

$$\begin{pmatrix} a & a & 1 \\ \alpha & -\alpha-1 & 1 \end{pmatrix} = (-1)^{a-\alpha}\sqrt{\frac{(a+\alpha+1)(a+\alpha+2)}{(2a+1)(2a+2)(2a+3)}}$$

$$\begin{pmatrix} a & b & c \\ -\alpha & -\beta & -\gamma \end{pmatrix} = (-1)^{a+b+c}\begin{pmatrix} a & b & c \\ \alpha & \beta & \gamma \end{pmatrix}$$

$$\begin{pmatrix} a & a & 1 \\ \alpha & -\alpha & 0 \end{pmatrix} = (-1)^{a-\alpha}\frac{\alpha}{\sqrt{a(a+1)(2a+1)}}$$

$$\begin{pmatrix} 1 & a & a \\ 0 & -\alpha & \alpha \end{pmatrix} = (-1)^{a-\alpha+1}\frac{\alpha}{\sqrt{a(a+1)(2a+1)}}$$

$$\begin{pmatrix} a & a & 2 \\ \alpha & -\alpha & 0 \end{pmatrix} = (-1)^{a-\alpha}\frac{3\alpha^2-a(a+1)}{\sqrt{a(a+1)(2a+3)(2a+1)(2a-1)}}$$

$$\begin{pmatrix} a & a & 2 \\ \alpha & -\alpha-1 & 1 \end{pmatrix} = (-1)^{a-\alpha}(2\alpha+1)\sqrt{\frac{3(a-\alpha)(a+\alpha+1)}{a(2a+3)(2a+2)(2a+1)(2a-1)}}$$

Figure M A few of the many equations useful in determination of specific Wigner 3j coefficients.

10.12Examples

Rank1_SpinT.cc

Rank 1 Spin Tensors

```
/* Rank1_SpinT.cc ************************************-*-c++-*-
**                                                   **
**                                        Example program for the GAMMA Library**
**                                                   **
** This program computes the standardized rank 1 spin tensor**
** tensor provided by GAMMA. The 4 tensor components are**
** output in FrameMaker MMF format for incorporation into doc-**
** uments as well as to standard output for viewing. These are**
** scaled appropriately so that the spin operators (tensor**
** components) have elements which are integer.**
**                                                   **
*****************************************************************/

#include <gamma.h>

void CompOut(matrix& SOp, int l, int m, complex& fact)

  {
  cout << "\n"
    << "\n\t 1"
    << "\n\t" << dtoa(Re(fact), 'f', 10, 2) << " * T"
    << "\n\t " << l << "," << m
    << "\n" << SOp;
  return;
  }


main (int argc, char* argv[])

  {
  cout << "\n\n\t\tGAMMA Rank 1 Spin Tensor Components\n\n";
  sys_dynamic dsys(1);                    // Set a 1 spin system
  spin_T T = T1(dsys, 0);                 // Get rank 1 spin tensor
  matrix SOp;                             // Working spin operator
  complex fact = 1.0;                     // Set scaling factor
  SOp = fact*matrix(T.component(0,0));    // Get 00 component
  CompOut(SOp, 0, 0, fact);               // Output to screen
  FM_Matrix("T00.mmf", SOp);              // Output T00 to FrameMaker
  fact = 2.0;                             // Set scaling factor
  SOp = fact*matrix(T.component(1,0));    // Get 10 component
  CompOut(SOp, 1, 0, fact);               // Output to screen
  FM_Matrix("T10.mmf", SOp);              // Output T10 to FrameMaker
  fact = 2.0/sqrt(2.0);                   // Set scaling factor
  SOp = fact*matrix(T.component(1,1));    // Get 11 component
  CompOut(SOp, 1, 1, fact);               // Output to screen
  FM_Matrix("T11.mmf", SOp);              // Output T11 to FrameMaker
  SOp = fact*matrix(T.component(1,-1));   // Get 1-1 component
  CompOut(SOp, 1, -1, fact);              // Output to screen
```

```
FM_Matrix("T1m1.mmf", SOp);          // Output T1-1 to FrameMaker
cout << "\n\n";                      // Keep screen nice
}
```

Rank2_SpinT.cc

Rank 2 Spin Tensors

```
/* Rank2_SpinT.cc ************************************-*-c++-*-
**                                  **
**         Example program for the GAMMA Library**
**                                  **
** This program computes the standardized rank 2 spin tensor **
** provided by GAMMA. The nine tensor components are output in**
** FrameMaker MMF format for incorporation into documents as**
** well as to standard output for viewing. These are scaled**
** appropriately so that the spin operators (tensor components)**
** have elements which are integer.**
**                                  **
*************************************************************/

#include <gamma.h>

void CompOut(matrix& SOp, int l, int m, complex& fact)

 {
 cout << "\n"
 << "\n\t\t 2"
 << "\n\t\t" << dtoa(Re(fact), 'f', 10, 2) << " * T"
 << "\n\t\t " << l << "," << m
 << "\n" << SOp;
 return;
 }


main (int argc, char* argv[])

 {
 cout << "\n\n\t\tGAMMA Rank 2 Spin Tensor Components\n\n";
 sys_dynamic dsys(2);                 // Set a 2 spin system
 spin_T T = T2(dsys, 0, 1);           // Get full rank 2 spin tensor
 matrix SOp;                          // Working spin operator
 complex fact = -4.0*sqrt(3.0);       // Set scaling factor
 SOp = fact*matrix(T.component(0,0)); // Get 00 component
 CompOut(SOp, 0, 0, fact);            // Output to screen
 FM_Matrix("T00.mmf", SOp);           // Output T00 to FrameMaker
 fact = -2.0*sqrt(2.0);               // Set scaling factor
 SOp = fact*matrix(T.component(1,0)); // Get 10 component
 CompOut(SOp, 1, 0, fact);            // Output to screen
 FM_Matrix("T10.mmf", SOp);           // Output T10 to FrameMaker
 fact = 4.0;                          // Set scaling factor
 SOp = fact*matrix(T.component(1,1)); // Get 11 component
```

```
CompOut(SOp, 1, 1, fact);                // Output to screen
FM_Matrix("T11.mmf", SOp);               // Output T11 to FrameMaker
SOp = fact*matrix(T.component(1,-1));    // Get 1-1 component
CompOut(SOp, 1, -1, fact);               // Output to screen
FM_Matrix("T1m1.mmf", SOp);              // Output T1-1 to FrameMaker
fact = 2.0*sqrt(6.0);                    // Set scaling factor
SOp = fact*matrix(T.component(2,0));     // Get 20 component
CompOut(SOp, 2, 0, fact);                // Output to screen
FM_Matrix("T20.mmf", SOp);               // Output T20 to FrameMaker
fact = 4.0;                              // Set scaling factor
SOp = fact*matrix(T.component(2,1));     // Get 21 component
CompOut(SOp, 2, 1, fact);                // Output to screen
FM_Matrix("T21.mmf", SOp);               // Output T21 to FrameMaker
SOp = fact*matrix(T.component(2,-1));    // Get 2-1 component
CompOut(SOp, 2, -1, fact);               // Output to screen
FM_Matrix("T2m1.mmf", SOp);              // Output T2-1 to FrameMaker
fact = 2.0;                              // Set scaling factor
SOp = fact*matrix(T.component(2,2));     // Get 22 component
CompOut(SOp, 2, 2, fact);                // Output to screen
FM_Matrix("T22.mmf", SOp);               // Output T22 to FrameMaker
SOp = fact*matrix(T.component(2,-2));    // Get 2-2 component
CompOut(SOp, 2, -2, fact);               // Output to screen
FM_Matrix("T2m2.mmf", SOp)               ;// Output T2-2 to FrameMaker
cout << "\n\n";                          // Keep screen nice
}
```

XProd_SpinT.cc

Tensor Products

```
/* XProd_SpinT.cc ************************************-*-c++-*-
**                                  **
**          Example program for the GAMMA Library**
**                                  **
** This program is used to test the tensor product routine in**
** the GAMMA spin tensor class, spin_T. It first computes two**
** standardized rank 1 spin tensors provided by GAMMA and then**
** generates a rank 2 (two spin) tensor. The tensors are output**
** to standard I/O unscaled.      **
**                                  **
*****************************************************************/

#include <gamma.h>

main (int argc, char* argv[])

{
cout << "\n\n\t\tGAMMA Spin Tensor Product Check\n\n";
sys_dynamic dsys(2);                     // Set a 1 spin system
spin_T T10 = T1(dsys, 0);                // Rank 1 spin tensor, 1st spin
spin_T T11 = T1(dsys, 1);                // Rank 1 spin tensor, 2nd spin
spin_T T201 = T_mult(T10,T11);           // Rank 2 from product
cout << "\n\n\tRank 1 Spin Tensor for Spin 1\n" << T10;
cout << "\n\n\tRank 1 Spin Tensor for Spin 2\n" << T11;
```

```
cout << "\n\n\tRank 2 Spin Tensor Product from Spins 1 & 2\n" << T201;
cout << "\n\n";                          // Keep screen nice
}
```

Rank2SS_SpinT.cc

Rank 2 Space/Spin Tensors

```
/* Rank2SS_SpinT.cc ************************************-*-c++-*-
**                                    **
**         Example program for the GAMMA Library**
**                                    **
** This program computes the standardized rank 2 space/spin**
** tensor provided by GAMMA. The nine tensor components are**
** output in FrameMaker MMF format for incorporation into doc-**
** uments as well as to standard output for viewing. These are**
** scaled appropriately so that the spin operators (tensor**
** components) have elements which are integer.**
**                                    **
*************************************************************/

#include <gamma.h>

void CompOut(matrix& SOp, int l, int m, complex& fact)

{
cout << "\n"
  << "\n\t 2"
  << "\n\t" << dtoa(Re(fact), 'f', 10, 2) << " * T"
  << "\n\t " << l << "," << m
  << "\n" << SOp;
return;
}


main (int argc, char* argv[])

{
cout << "\n\n\t\tGAMMA Rank 2 Space/Spin Tensor Components\n\n";
sys_dynamic dsys(1);                   // Set a 1 spin system
coord B(0,0,1);                        // Set z-axis vector
spin_T T = T2(dsys, 0, B);             // Get rank 2 space/spin tensor
matrix SOp;                            // Working spin operator
complex fact = -2.0*sqrt(3.0);         // Set scaling factor
SOp = fact*matrix(T.component(0,0));   // Get 00 component
CompOut(SOp, 0, 0, fact);              // Output to screen
FM_Matrix("T00.mmf", SOp);             // Output T00 to FrameMaker
fact = 1.0;                            // Set scaling factor
SOp = fact*matrix(T.component(1,0));   // Get 10 component
CompOut(SOp, 1, 0, fact);              // Output to screen
FM_Matrix("T10.mmf", SOp);             // Output T10 to FrameMaker
fact = -2.0;                           // Set scaling factor
SOp = fact*matrix(T.component(1,1));   // Get 11 component
```

```
CompOut(SOp, 1, 1, fact);                 // Output to screen
FM_Matrix("T11.mmf", SOp);                // Output T11 to FrameMaker
SOp = fact*matrix(T.component(1,-1));     // Get 1-1 component
CompOut(SOp, 1, -1, fact);                // Output to screen
FM_Matrix("T1m1.mmf", SOp);               // Output T1-1 to FrameMaker
fact = 2.0*sqrt(3.0/2.0);                 // Set scaling factor
SOp = fact*matrix(T.component(2,0));      // Get 20 component
CompOut(SOp, 2, 0, fact);                 // Output to screen
FM_Matrix("T20.mmf", SOp);                // Output T20 to FrameMaker
fact = 2.0;                               // Set scaling factor
SOp = fact*matrix(T.component(2,1));      // Get 21 component
CompOut(SOp, 2, 1, fact);                 // Output to screen
FM_Matrix("T21.mmf", SOp);                // Output T21 to FrameMaker
SOp = fact*matrix(T.component(2,-1));     // Get 2-1 component
CompOut(SOp, 2, -1, fact);                // Output to screen
FM_Matrix("T2m1.mmf", SOp);               // Output T2-1 to FrameMaker
fact = 1.0;                               // Set scaling factor
SOp = fact*matrix(T.component(2,2));      // Get 22 component
CompOut(SOp, 2, 2, fact);                 // Output to screen
FM_Matrix("T22.mmf", SOp);                // Output T22 to FrameMaker
SOp = fact*matrix(T.component(2,-2));     // Get 2-2 component
CompOut(SOp, 2, -2, fact);                // Output to screen
FM_Matrix("T2m2.mmf", SOp);               // Output T2-2 to FrameMaker
cout << "\n\n";                           // Keep screen nice
}
```