*1 Introduction ........................................................................ 5*

*2 Single Spin Operators ........................................................ 6*

2.1 Overview ........................................................................ 6
2.2 Available Functions ........................................................ 6
2.3 Basic Functions ................................................................ 7
2.4 Description ...................................................................... 14

*3 Base Spin System ............................................................... 26*

3.1 Overview ........................................................................ 26
3.2 Available Functions ........................................................ 26
3.3 Algebraic Operators ........................................................ 28
3.4 Basic Functions ................................................................ 31
3.5 Spin Flags Functions ....................................................... 45
3.6 Input/Output ................................................................... 48
3.7 Description ...................................................................... 51
3.8 Parameter Files ................................................................ 52
3.9 Example Parameter Files ................................................. 53
3.10 Example Programs ......................................................... 54
3.11 List of Available Isotopes .............................................. 55

*4 Spin Operators .................................................................... 58*

4.1 Overview ........................................................................ 58
4.2 Available Spin Operator Functions .................................. 58
4.3 Routines .......................................................................... 60
4.4 Single-Spin Spin Operators ............................................. 66
4.5 Multiple-Spin Spin Operators .......................................... 78
4.6 Rotation Operators .......................................................... 88
4.7 Rotated Spin Operators ................................................... 102
4.8 Spin Operator General Functions .................................... 104
4.9 Description ...................................................................... 106
4.10 Implicit Spin Operators .................................................. 115

*5 Class Basis ......................................................................... 116*

5.1 Overview ........................................................................ 116
5.2 Available Basis Functions ................................................ 117
5.3 Routines .......................................................................... 118
5.4 Description ...................................................................... 124
5.5 Implementation ............................................................... 125

*6 Class Operator (gen_op) ..................................................... 126*

6.1 Overview ........................................................................ 126
6.2 Available Operator Functions .......................................... 126
6.3 Arithmetic Operators ...................................................... 128
6.4 Complex Functions ......................................................... 137

6.5 Operator Internal Access ........ 141
6.6 Basis Manipulations ........ 143
6.7 Representation Manipulations ........ 146
6.8 Operator I/O Functions ........ 148
6.9 Description ........ 149

***7 Class Spin System ........ 155***

7.1 Overview ........ 155
7.2 Available Spin System Functions ........ 155
7.3 Spin System Algebraic Functions ........ 157
7.4 Spin System Basic Functions ........ 158
7.5 Spin System Associated Functions ........ 165
7.6 Spin System I/O Functions ........ 168
7.7 Description ........ 170
7.8 Spin System Parameter Files ........ 171
7.9 Example Spin System Parameter Files ........ 173
7.10 Example Source Codes ........ 174

***8 Hamiltonians ........ 175***

8.1 Overview ........ 175
8.2 Available Hamiltonian Functions ........ 175
8.3 Covered Hamiltonian Theory ........ 175
8.4 Hamiltonian Figures ........ 176
8.5 Hamiltonian Example Programs ........ 176
8.6 Routines ........ 177
8.7 Description ........ 189
8.8 Example Source Codes ........ 261

***9 Ideal Pulses ........ 271***

9.1 Overview ........ 271
9.2 Available Functions ........ 271
9.3 Figures & Tables ........ 271
9.4 Pulse Routines ........ 272
9.5 Propagator Routines ........ 278
9.6 Description ........ 284
9.7 Source Codes ........ 287

***10 Rectangular Pulses ........ 289***

10.1 Overview ........ 289
10.2 Rectangular Pulse Functions ........ 289
10.3 Rectangular Pulse Figures ........ 289
10.4 Rectangular Pulse Theory ........ 289
10.5 Rectangular Pulse Example Programs ........ 289
10.6 Routines ........ 290
10.7 Description ........ 301

10.8 Example Source Codes .......... 310

*11 Shaped Pulses .......... 312*

11.1 Overview .......... 312
11.2 Shaped Pulse Functions .......... 312
11.3 Routines .......... 312
11.4 Description .......... 319
11.5 Chapter Source Codes .......... 319

*12 Evolution .......... 320*

12.1 Overview .......... 320
12.2 Evolution Functions .......... 320
12.3 Routines .......... 320
12.4 Description .......... 324

# GAMMA

# Hilbert Space Module

$$\hat{\hat{\Gamma}}$$

Author: Dr. Scott A. Smith, Tilo Levante

Date: March 14, 2000

# 1 Introduction

The Hilbert space module in GAMMA lays down the foundation for performing magnetic resonance simulations using quantum mechanics. At the core are three features. First, a complete set of single spin operators are provided as small dimensional arrays. Second, the spin system class(es) allow users to set up a group of spins that define a composite spin Hilbert space. Third, the module provides a host of spin operators defined in the composite space of a provided spin system. These are built up from direct produces of single spin operators and thus general for any spin systems.

Extending these three abilities, GAMMA provides simple functions for constructing spin base Hamiltonians and functions for density operator manipulations. Simple pulse, delay, and acquisition functionality is set up in this module.

# 2 Single Spin Operators

## 2.1 Overview

The GAMMA module ***SpinOpSng*** contains routines to handle spin operators associated with a single particle. These serve as a front end to class ***spin_op*** which treats spin operators associated with a system of particles. Each ***SpinOpSng*** exists in the small Hilbert space of its particle whereas each ***spin_op*** will exist in a larger composite Hilbert space produced from all its particles. The latter is formed from the tensor product of ***SpinOpSng*** for each of its particles.

Normally GAMMA users do NOT deal with single spin operators. Rather, they deal with spin operators (class ***spin_op)*** defined over spin systems. This is much more general and produces equivalent results when the spin system only contains a single spin particle.

## 2.2 Available Functions

**Basic Functions**

| | | |
|---|---|---|
| Ie | - Identity single spin operator | page 7 |
| Ix | - Ix single spin operator | page 8 |
| Iy | - Iy single spin operator | page 9 |
| Iz | - Iz single spin operator | page 10 |
| Ip | - I+ single spin operator | page 11 |
| Im | - I- single spin operator | page 12 |
| Raxis | - Single spin rotation matrix | page 13 |

# 2.3 Basic Functions

## 2.3.1 Ie

### Usage:

#include <HSLib/SpinOpSng.h>
matrix Ie (int qn)

### Description:

The function ***Ie*** is used to create asingle spin identity spin operator. There function takes the dimension of the single spin Hilbert space, ***qn***, as input where ***qn = (2I+1)*** and ***I*** is the spin quantum number of the spin for which the operator applies.

### Return Value:

An identity matrix is returned.

### Example:

```
#include <gamma.h>
main ()
  {
  matrix IE = Ie(2);                          // The Ie spin operator for a single spin 1/2 particle
  }
```

**See Also:** Ix, Iy, Iz, Ip, Im

### Mathematical Basis:

There are four operators which provide a basis for spin angular momentum, the set **{$\mathbf{I_e}$, $\mathbf{I_x}$, $\mathbf{I_y}$, $\mathbf{I_z}$}**. Although $\boldsymbol{I_e}$ is trivial (being simply the identity matrix) its provision is essential in order to complete the basis mentioned above. The matrix representation of $\boldsymbol{I_e}$ for a single spin is efficiently expressed in the dimension of the Hilbert space of the spin itself, spanning 2($I_i$+1). Examples are shown below for particles with spin 1/2, 1, and 3/2 respectively.

$$\boldsymbol{I}_e\left(I = \frac{1}{2}\right) = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \qquad \boldsymbol{I}_e(I = 1) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad \boldsymbol{I}_e\left(I = \frac{3}{2}\right) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The $\boldsymbol{I_e}$ operators are used most frequently in the expansion of other spin operators into their composite Hilbert space forms through use of these direct products.

## 2.3.2 Ix

### Usage:

```
#include <HSLib/SpinOpSng.h>
matrix Ix(Int qn)
```

### Description:

The function ***Ix*** is used to create a single spin operator for the x-component of spin angular momentum. There function takes the dimension of the single spin Hilbert space, ***qn***, as input where ***qn = (2I+1)*** and ***I*** is the spin quantum number of the spin for which the operator applies.

### Return Value:

A matrix (h_matrix) is returned.

### Example:

```
#include <gamma.h>
main ()
  {
  matrix IX = Ix(3);                        // The Ix spin operator for a single spin 1 particle
  }
```

**See Also:** Ie, Iy, Iz, Ip, Im

### Mathematical Basis:

There are four operators which provide a basis for spin angular momentum, the set **{$\mathbf{I_e}$, $\mathbf{I_x}$, $\mathbf{I_y}$, $\mathbf{I_z}$}**. The matrix representation of $\boldsymbol{I_x}$ for a single spin is efficiently expressed in the dimension of the Hilbert space of the spin itself, spanning $2(I_i+1)$. Examples are shown below for particles with spin 1/2, 1, and 3/2 respectively.

$$\boldsymbol{I}_x(I = 1/2) = \frac{1}{2}\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \qquad \boldsymbol{I}_x(I = 1) = \frac{1}{\sqrt{2}}\begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \qquad \boldsymbol{I}_x\left(I = \frac{3}{2}\right) = \frac{1}{2}\begin{bmatrix} 0 & \sqrt{3} & 0 & 0 \\ \sqrt{3} & 0 & 2 & 0 \\ 0 & 2 & 0 & \sqrt{3} \\ 0 & 0 & \sqrt{3} & 0 \end{bmatrix}$$

The $\boldsymbol{I_x}$ operators are used most frequently in the expansion of other spin operators into their composite Hilbert space forms through use of these direct products. In general, the matrix elements of the operator are given by[1]

$$\langle m_i|\boldsymbol{I}_{ix}|n_i\rangle = \frac{1}{2}[I_i(I_i+1) - m_i(m_i \pm 1)]^{1/2}\delta_{m_i,\, n_i \pm 1}$$

Other important relationships involving this operator are the following.

$$\boldsymbol{I}_x = \frac{1}{2}(\boldsymbol{I}_+ + \boldsymbol{I}_-) \qquad [\boldsymbol{I}_x, \boldsymbol{I}_y] = i\boldsymbol{I}_z \qquad [\boldsymbol{I}_x, \boldsymbol{I}_z] = -i\boldsymbol{I}_y$$

[1] Goldman, page 73, equation (3.93).

## 2.3.3 Iy

### Usage:

```
#include <HSLib/SpinOpSng.h>
matrix Iy(int qn)
```

### Description:

The function ***Iy*** is used to create a single spin operator for the y-component of spin angular momentum. There function takes the dimension of the single spin Hilbert space, ***qn***, as input where ***qn = (2I+1)*** and ***I*** is the spin quantum number of the spin for which the operator applies.

### Return Value:

A matrix (h_matrix) is returned.

### Example:

```
#include <gamma.h>
main ()
  { matrix IY = Iy(2); }                    // The Iy spin operator for a single spin 1/2 particle
```

**See Also:** Ie, Ix, Iz, Ip, Im

### Mathematical Basis:

There are four operators which provide a basis for spin angular momentum, the set **{I$_e$, I$_x$, I$_y$, I$_z$}**. The matrix representation of ***Iy*** for a single spin is efficiently expressed in the dimension of the Hilbert space of the spin itself, spanning 2(I$_i$+1). Examples are shown below for particles with spin 1/2, 1, and 3/2 respectively[1].

$$\boldsymbol{I}_y\left(I = \frac{1}{2}\right) = \frac{i}{2}\begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \qquad \boldsymbol{I}_y(I = 1) = \frac{i}{\sqrt{2}}\begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix} \qquad \boldsymbol{I}_y\left(I = \frac{3}{2}\right) = \frac{i}{2}\begin{bmatrix} 0 & -\sqrt{3} & 0 & 0 \\ \sqrt{3} & 0 & -2 & 0 \\ 0 & 2 & 0 & -\sqrt{3} \\ 0 & 0 & \sqrt{3} & 0 \end{bmatrix}$$

The ***Iy*** operators are used most frequently in the expansion of other spin operators into their composite Hilbert space forms through use of these direct products. In general, the matrix elements of the operator are given by[2]

$$\langle m_i|\boldsymbol{I}_{iy}|n_i\rangle = \left(\pm\frac{i}{2}\right)[I_i(I_i + 1) - m_i(m_i \pm 1)]^{1/2}\delta_{m_i,\, n_i \pm 1}$$

Other important relationships involving this operator are the following.

$$\boldsymbol{I}_y = \frac{i}{2}(\boldsymbol{I}_+ - \boldsymbol{I}_-) \qquad [\boldsymbol{I}_y,\boldsymbol{I}_x] = -i\boldsymbol{I}_z \qquad [\boldsymbol{I}_y,\boldsymbol{I}_z] = i\boldsymbol{I}_x$$

[1] These leave out the units of hbar, see Schiff, page 203.

[2] Goldman, page 73, equation (3.93).

## 2.3.4 Iz

### Usage:

#include <HSLib/SpinOpSng.h>
matrix Iz(int qn)

### Description:

The function ***Iz*** is used to create a single spin operator for the z-component of spin angular momentum. There function takes the dimension of the single spin Hilbert space, ***qn***, as input where ***qn = (2I+1)*** and ***I*** is the spin quantum number of the spin for which the operator applies.

### Return Value:

A diagonal matrix (d_matix) is returned.

### Example(s):

```
#include <gamma.h>
main ()
  { matrix IZ = Iz(2); }                    // The Iz spin operator for a single spin 1/2 particle
```

**See Also:** Ie, Ix, Iy, Ip, Im

### Mathematical Basis:

There are four operators which provide a basis for spin angular momentum, the set $\mathbf{\{I_e, I_x, I_y, I_z\}}$. The matrix representation of $\mathbf{I_z}$ for a single spin is efficiently expressed in the dimension of the Hilbert space of the spin itself, spanning $2(I_i+1)$. Examples are shown below for particles with spin 1/2, 1, and 3/2 respectively[1].

$$\boldsymbol{I}_{iz}(I_i = 1/2) = \frac{1}{2}\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \qquad \boldsymbol{I}_{iz}(I_i = 1) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & -1 \end{bmatrix} \qquad \boldsymbol{I}_{iz}\left(I_i = \frac{3}{2}\right) = \frac{1}{2}\begin{bmatrix} 3 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & -3 \end{bmatrix}$$

The $\mathbf{I_z}$ operators are used most frequently in the expansion of other spin operators into their composite Hilbert space forms through use of these direct products. In general, the matrix elements of the operator are given by[2]

$$\langle n_i|\boldsymbol{I}_{iz}|m_i\rangle = m\delta_{m_i n_i} = [I(1-\alpha)]\delta_{m_i n_i}$$

Other important relationships involving this operator are the following.

$$[\boldsymbol{I}_{iz}, \boldsymbol{I}_{ix}] = i\boldsymbol{I}_{iy} \qquad [\boldsymbol{I}_{iz}, \boldsymbol{I}_{i+}] = \boldsymbol{I}_{i+} \qquad [\boldsymbol{I}_{iz}, \boldsymbol{I}_{i-}] = \boldsymbol{I}_{i-}$$

[1] These leave out the units of hbar, see Schiff, page 203.

[2] Goldman, page 73, equation (3.93).

## 2.3.5 Ip

### Usage:

```
#include <HSLib/SpinOpSng.h>
matrix Ip(int qn)
```

### Description:

The function ***Ip*** is used to create a single spin s raising operator $\mathbf{I}_+$. There function takes the dimension of the single spin Hilbert space, ***qn***, as input where ***qn = (2I+1)*** and ***I*** is the spin quantum number of the spin for which the operator applies.

### Return Value:

A matrix (n_matrix) is returned.

### Example(s):

```
#include <gamma.h>
main ()
  { matrix IP = Ip(2); }                          // The I+ spin operator for a single spin 1/2 particle
```

**See Also:** Ie, Iy, Iz, Ix, Im

### Mathematical Basis:

The raising operator $\mathbf{I}_+$ is defined to work on state |***I,m***> as[1]

$$\boldsymbol{I}_+|I, m\rangle = [I(I+1) - m(m+1)]^{1/2}|I, m+1\rangle,$$

having matrix elements

$$\langle I, m'|\boldsymbol{I}_+|I, m\rangle = [I(I+1) - m(m+1)]^{1/2}\delta_{m+1, m'}.$$

The matrix representation of $\mathbf{I}_+$ for a single spin i**,** $\mathbf{I}_{i+}$**,** is efficiently expressed in the dimension of the Hilbert space of the spin itself, spanning $2(I_i+1)$. Examples are shown below for a spin 1/2 and a spin 1 particle

$$\boldsymbol{I}_{i+}\left(I_i = \frac{1}{2}\right) = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \qquad \boldsymbol{I}_{i+}(I_i = 1) = \sqrt{2}\begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix} \qquad \boldsymbol{I}_{i+}\left(I_i = \frac{3}{2}\right) = \begin{bmatrix} 0 & \sqrt{3} & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & \sqrt{3} \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Other useful relationships involving this operator are the following.

$$\boldsymbol{I}_{i+} = (\boldsymbol{I}_{ix} + i\boldsymbol{I}_{iy}) \qquad [\boldsymbol{I}_{i+}, \boldsymbol{I}_{iz}] = -\boldsymbol{I}_{i+} \qquad [\boldsymbol{I}_{i+}, \boldsymbol{I}_{i-}] = 2\boldsymbol{I}_{iz} \qquad \boldsymbol{I}_{i+} = (\boldsymbol{I}_{i-})^{\dagger}$$

[1] Schiff, pg. 202, equation (27.25).

## 2.3.6 Im

### Usage:

```
#include <HSLib/SpinOpSng.h>
matrix Im(int qn)
```

### Description:

The function ***Im*** is used to create a single spin lowering operator **I**$_-$. There function takes the dimension of the single spin Hilbert space, ***qn***, as input where ***qn = (2I+1)*** and ***I*** is the spin quantum number of the spin for which the operator applies.

### Return Value:

A matrix (n_matrix) is returned.

### Example(s):

```
#include <gamma.h>
main ()
  { matrix IM = Im(2); }                    // The I- spin operator for a single spin 1/2 particle
```

**See Also:** Ie, Ix, Iy, Iz, Ip

### Mathematical Basis:

The lowering operator **I**$_-$ is defined to work on state **|*I,m*>** as[1]

$$\boldsymbol{I}_-|I, m\rangle = [I(I+1) - m(m-1)]^{1/2}|I, m-1\rangle,$$

having matrix elements

$$\langle I, m'|\boldsymbol{I}_-|I, m\rangle = [I(I+1) - m(m-1)]^{1/2}\delta_{m-1, m'}.$$

The matrix representation of **I**$_-$ for a single spin i**, I**$_-$**,** is efficiently expressed in the dimension of the Hilbert space of the spin itself, spanning 2(I$_i$+1). Examples are shown below for particles with spin angular momentum 1/2, 1, and 3/2 respectively.

$$\boldsymbol{I}_{i\text{-}}\left(I_i = \frac{1}{2}\right) = \begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix} \qquad \boldsymbol{I}_{i\text{-}}(I_i = 1) = \sqrt{2}\begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \qquad \boldsymbol{I}_{i\text{-}}\left(I_i = \frac{3}{2}\right) = \begin{bmatrix} 0 & 0 & 0 & 0 \\ \sqrt{3} & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & \sqrt{3} & 0 \end{bmatrix}$$

Other useful relationships involving this operator are the following.

$$\boldsymbol{I}_{i\text{-}} = (\boldsymbol{I}_{ix} - i\boldsymbol{I}_{iy}) \qquad \boldsymbol{I}_{i\text{-}} = (\boldsymbol{I}_{i+})^{\dagger} \qquad [\boldsymbol{I}_{i\text{-}}, \boldsymbol{I}_{iz}] = \boldsymbol{I}_{i\text{-}} \qquad [\boldsymbol{I}_{i+}, \boldsymbol{I}_{i\text{-}}] = 2\boldsymbol{I}_{iz}$$

[1] Schiff, pg. 202, equation (27.25).

### 2.3.7 Raxis

**Usage:**

```
#include <HSLib/SpinOpSng.h>
matrix Raxis(int qn, double beta, char axis)
```

**Description:**

The function ***Raxis*** is used to create a single spin rotation operator, $\boldsymbol{R}_u(\beta)$ , for rotating spin angular momentum about the $\boldsymbol{u}$ -axis by the angle β. This is defined by the formula

$$\boldsymbol{R}_u(\beta) = exp(-i\beta I_u)$$

where $\boldsymbol{I_u}$ is an appropriate single spin angular momentum operator and $\boldsymbol{u}$ one of the three Cartesian axes.

**Return Value:**

A matrix is returned.

**Example:**

```
#include <gamma.h>
main ()
  { matrix Rx90 = Raxis(2, 90.0, ‘x’); }        // Rotation op. for a spin 1/2 particle, 90 deg. about x
```

**See Also:**

**Mathematical Basis:**

The formulae used for a single spin 1/2 particle are[1] (using $C\beta = \cos(\beta/2) \qquad S\beta = \sin(\beta/2)$)

$$\boldsymbol{R}_{ix}^{(1/2)}(\beta) = \begin{bmatrix} C\beta & -iS\beta \\ -iS\beta & C\beta \end{bmatrix} \qquad \boldsymbol{R}_y(\beta) = \begin{bmatrix} C\beta & -S\beta \\ S\beta & C\beta \end{bmatrix} \qquad \boldsymbol{R}_z(\beta) = \begin{bmatrix} e^{-i\beta/2} & 0 \\ 0 & e^{i\beta/2} \end{bmatrix}$$

Unfortunately, there is currently no similar simple formula for a spin with I > 1/2 and these are computed by actual exponentiation of the operator. A few specific rotation matrices one will obtain from this function are as follows.

$$\boldsymbol{R}_{ix}^{(1/2)}\left(\frac{\pi}{3}\right) = \begin{bmatrix} 0.866 & -0.5i \\ -0.5i & 0.866 \end{bmatrix} \qquad \boldsymbol{R}_{ix}^{(1/2)}\left(\frac{\pi}{2}\right) = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 & -i \\ -i & 1 \end{bmatrix} \qquad \boldsymbol{R}_{ix}^{(1/2)}(\pi) = \begin{bmatrix} 0 & -i \\ -i & 0 \end{bmatrix}$$

$$\boldsymbol{R}_{iy}^{(1/2)}\left(\frac{\pi}{3}\right) = \begin{bmatrix} 0.866 & -0.5 \\ 0.5 & 0.866 \end{bmatrix} \qquad \boldsymbol{R}_{iy}^{(1/2)}\left(\frac{\pi}{2}\right) = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 & -1 \\ 1 & 1 \end{bmatrix} \qquad \boldsymbol{R}_{iy}^{(1/2)}(\pi) = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}$$

$$\boldsymbol{R}_{iz}^{(1/2)}\left(\frac{\pi}{3}\right) = \begin{bmatrix} a^* & 0 \\ 0 & a \end{bmatrix} \qquad \boldsymbol{R}_{iz}^{(1/2)}\left(\frac{\pi}{2}\right) = \frac{1}{\sqrt{2}}\begin{bmatrix} 1+i & 0 \\ 0 & 1+i \end{bmatrix} \qquad \boldsymbol{R}_{iz}^{(1/2)}(\pi) = \begin{bmatrix} -i & 0 \\ 0 & i \end{bmatrix}$$

$$a = 0.866 + 0.5i$$

[1] See Ernst, Bodenhausen, and Wokaun, page 405, equation (8.1.4).

## 2.4 Description

The GAMMA module ***SpinOpSng*** provides all the common single spin angular momentum operators. These are used internally to provide the class ***spin_op*** with the core elements needed for construction of spin operators defined over systems containing multiple spins. Normally, GAMMA users do not employ these single spin operator functions explicitly in programs. Rather, spin operators defined over the Hilbert space of a spin system are used. These are more general and will handle any number of spins in the system and in any combination. When the system contains only one spin then such spin operators will be equivalent to the single spin operators returned by the functions of this section.

Each spin operator can be presented is in terms of single spin operators. In turn, each single spin operator in this context is stored in the spin Hilbert space of dimension of 2*I*+1, not in the full spin system Hilbert space. The full Hilbert space representation is obtained from these single spin matrices by taking direct products.

$$\boldsymbol{SOp} = \boldsymbol{SOp}(1) \otimes \boldsymbol{SOp}(2) \otimes \boldsymbol{SOp}(3) \otimes \ldots \otimes \boldsymbol{SOp}(n) \qquad (2\text{-}1)$$

In this equation, ***SOp(i)*** is the single spin operator acting on spin i. All spins which are active in ***SOp*** are included in the product and each single spin operator here spans its own Hilbert Space.

Although not generally possible for any arbitrary spin operators[1], use of this operator description can result in both computational and memory savings. The memory savings can be made evident by consideration of the spin operator $\mathbf{I}_{1x}$ in a three spin system with spins having quantum numbers of 1/2, 1, and 1/2 respectively. The full Hilbert space of this spin system is 12 resulting from the product of the individual spin sub-spaces of 2, 3, and 2. The operator then looks like

$$\boldsymbol{I}_{1x} = \begin{bmatrix} 12 \times 12 \\ Array \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \otimes \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \otimes \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} . \qquad (2\text{-}2)$$

or equivalently

$$\boldsymbol{I}_{1x} = \boldsymbol{I}_{1x} \otimes \boldsymbol{I}_{2e} \otimes \boldsymbol{I}_{3e} \qquad \mathbf{(2\text{-}3)}$$

For this simple three spin system, the full Hilbert space representation is a 12 x 12 array whereas by the individual spin matrices it is significantly smaller. GAMMA wisely stores only the diagonal elements in a diagonal array, thus internally storing only 9 elements rather than 144 for this array. Computations occur along a similar vein, when GAMMA can work with this sub-space representations it will do so preferentially, hence running significantly faster than when working with the expanded form.

[1] The question immediately arises as to when a spin operator (or any operator) can be expressed in terms of direct products. This will be true when the operator can be broken up into products of mutually commuting operators, such as the set $\{\mathbf{I}_{ix}\}$ for a spin system which all must commute as they act on different spins.

## 2.4.1 Single Spin Ix Operator

There are four Cartesian spin operators which provide a basis for spin angular momentum, the set **{$\mathbf{I_e}$, $\mathbf{I_x}$, $\mathbf{I_y}$, $\mathbf{I_z}$}**. The $\mathbf{I_x}$ operators are used most frequently in the expansion of other spin operators into their composite Hilbert space forms through use of these direct products. In general, the matrix elements of the operator are given by[1]

$$\langle m_i|\boldsymbol{I}_{ix}|n_i\rangle = \frac{1}{2}[I_i(I_i+1) - m_i(m_i \pm 1)]^{1/2}\delta_{m_i,\, n_i \pm 1}$$

Some important relationships involving this operator are

$$\boldsymbol{I}_x = \frac{1}{2}(\boldsymbol{I}_+ + \boldsymbol{I}_-) \qquad [\boldsymbol{I}_x,\boldsymbol{I}_y] = i\boldsymbol{I}_z \qquad [\boldsymbol{I}_x,\boldsymbol{I}_z] = -i\boldsymbol{I}_y$$

$\mathbf{I_x}$ acting on the simple basis states |α> and |β> produce

$$\boldsymbol{I}_x|\alpha\rangle = \frac{1}{2}(\boldsymbol{I}_+ + \boldsymbol{I}_-)|\alpha\rangle = \frac{1}{2}|\beta\rangle \qquad \boldsymbol{I}_x|\beta\rangle = \frac{1}{2}(\boldsymbol{I}_+ + \boldsymbol{I}_-)|\beta\rangle = \frac{1}{2}|\alpha\rangle \tag{2-4}$$

The acutal eigenkets of $\mathbf{I_x}$ for a spin 1/2 particle are given by[2]

$$|+_x\rangle = \frac{1}{\sqrt{2}}[|\alpha\rangle + |\beta\rangle] \qquad |-_x\rangle = \frac{1}{\sqrt{2}}[-|\alpha\rangle + |\beta\rangle] \tag{2-5}$$

as is easily obtained from rotations of the eigenkets of $\mathbf{I_z}$ about the y-axis by 90 degrees, that is,

$$\boldsymbol{R}_y^{(1/2)}\left(\frac{\pi}{2}\right)|\alpha\rangle = |+_x\rangle \qquad \boldsymbol{R}_y^{(1/2)}\left(\frac{\pi}{2}\right)|\beta\rangle = |-_x\rangle \tag{2-6}$$

[1] Goldman, page 73, equation (3.93).

[2] Goldman, pgs. 66-67.

## 2.4.2 Single Spin Iy Operator:

The $\mathbf{I_y}$ operators are used most frequently in the expansion of other spin operators into their composite Hilbert space forms through use of these direct products. In general, the matrix elements of the operator are given by[1]

$$\langle m_i|\boldsymbol{I}_{iy}|n_i\rangle = \left(\pm\frac{i}{2}\right)[I_i(I_i+1)-m_i(m_i\pm1)]^{1/2}\delta_{m_i,\, n_i\pm1} \tag{2-7}$$

Some important relationships involving this operator are the following.

$$\boldsymbol{I}_y = \frac{i}{2}(\boldsymbol{I}_+ - \boldsymbol{I}_-) \qquad [\boldsymbol{I}_y,\boldsymbol{I}_x] = -i\boldsymbol{I}_z \qquad [\boldsymbol{I}_y,\boldsymbol{I}_z] = i\boldsymbol{I}_x$$

$\mathbf{I_y}$ acting on the simple basis states |α> and |β> produce

$$\boldsymbol{I}_y|\alpha\rangle = -\frac{i}{2}(\boldsymbol{I}_+ - \boldsymbol{I}_-)|\alpha\rangle = \frac{i}{2}|\beta\rangle \qquad \boldsymbol{I}_y|\beta\rangle = -\frac{i}{2}(\boldsymbol{I}_+ - \boldsymbol{I}_-)|\beta\rangle = -\frac{i}{2}|\alpha\rangle \tag{2-8}$$

The acutal eigenkets of $\mathbf{I_y}$ for a spin 1/2 particle are given by[2]

$$|+_y\rangle = \frac{1}{\sqrt{2}}[|\alpha\rangle + i|\beta\rangle] \qquad |-_y\rangle = \frac{1}{\sqrt{2}}[i|\alpha\rangle - |\beta\rangle] \tag{2-9}$$

[1] Goldman, page 73, equation (3.94).
[2] Goldman, pg. 67, equation (3.68).

## 2.4.3 Single Spin Iz Operator:

The spin operator $\boldsymbol{I_{iz}}$ has the property that

$$\boldsymbol{I}_{iz}|m_i\rangle = m|m_i\rangle \tag{2-10}$$

where $|m_i\rangle$ is a basis function for spin i with spin quantum number ***m***. For a spin 1/2 particle the two possible spin states are |α> and |β>, and the following relationships hold.

$$\boldsymbol{I}_z|\alpha\rangle = \frac{1}{2}|\alpha\rangle \qquad \boldsymbol{I}_z|\beta\rangle = -\frac{1}{2}|\beta\rangle \tag{2-11}$$

From equation (2-10), the individual matrix elements of the operator are

$$\langle n_i|\boldsymbol{I}_{iz}|m_i\rangle = m\delta_{m_in_i} = [I(1-\alpha)]\delta_{m_in_i} \quad \text{where } \alpha \text{ spans } [1, 2(I+1)] \tag{2-12}$$

Other useful relationships involving $\boldsymbol{I}_{iz}$ are the following[1].

$$[\boldsymbol{I}_{iz}, \boldsymbol{I}_{ix}] = i\boldsymbol{I}_{iy} \qquad [\boldsymbol{I}_{iz}, \boldsymbol{I}_{i+}] = \boldsymbol{I}_{i+} \qquad [\boldsymbol{I}_{iz}, \boldsymbol{I}_{i-}] = \boldsymbol{I}_{i-} \tag{2-13}$$

Examples of $\boldsymbol{I_z}$ matrix representatins are shown below for particles with spin 1/2, 1, and 3/2 respectively[2].

$$\boldsymbol{I}_{iz}(I_i = 1/2) = \frac{1}{2}\begin{bmatrix}1 & 0\\0 & -1\end{bmatrix} \qquad \boldsymbol{I}_{iz}(I_i = 1) = \begin{bmatrix}1 & 0 & 0\\0 & 0 & 0\\0 & 0 & -1\end{bmatrix} \qquad \boldsymbol{I}_{iz}\left(I_i = \frac{3}{2}\right) = \frac{1}{2}\begin{bmatrix}3 & 0 & 0 & 0\\0 & 1 & 0 & 0\\0 & 0 & -1 & 0\\0 & 0 & 0 & -3\end{bmatrix}$$

That the equations (2-11) are satisfied for these matrix forms are now shown for the single spin case with I = 1/2 and with I = 1.

$$\boldsymbol{I}_{iz}|\alpha\rangle = \frac{1}{2}\begin{bmatrix}1 & 0\\0 & -1\end{bmatrix}\begin{bmatrix}1\\0\end{bmatrix} = \frac{1}{2}\begin{bmatrix}1\\0\end{bmatrix} = \frac{1}{2}|\alpha\rangle \qquad \boldsymbol{I}_{iz}|\beta\rangle = \frac{1}{2}\begin{bmatrix}1 & 0\\0 & -1\end{bmatrix}\begin{bmatrix}0\\1\end{bmatrix} = \frac{1}{2}\begin{bmatrix}0\\-1\end{bmatrix} = \left(-\frac{1}{2}\right)|\beta\rangle$$

$$\boldsymbol{I}_z|\alpha\rangle = \begin{bmatrix}1 & 0 & 0\\0 & 1 & 0\\0 & 0 & -1\end{bmatrix}\begin{bmatrix}1\\0\\0\end{bmatrix} = \begin{bmatrix}1\\0\\0\end{bmatrix} = 1|\alpha\rangle \qquad \boldsymbol{I}_z|\beta\rangle = \begin{bmatrix}1 & 0 & 0\\0 & 0 & 0\\0 & 0 & -1\end{bmatrix}\begin{bmatrix}0\\1\\0\end{bmatrix} = 0|\beta\rangle \qquad \boldsymbol{I}_z|\gamma\rangle = \begin{bmatrix}1 & 0 & 0\\0 & 1 & 0\\0 & 0 & -1\end{bmatrix}\begin{bmatrix}0\\0\\1\end{bmatrix} = \begin{bmatrix}0\\0\\-1\end{bmatrix} = -1|\gamma\rangle\,.$$

[1] See Goldman, p. 61-62.

[2] These leave out the units of hbar, see Schiff, page 203.

# 2.4.4 Single Spin I+ Operator

The raising operator $\boldsymbol{I}_+$ is defined to work on state **|*I,m*>** as[1]

$$\boldsymbol{I}_+|I, m\rangle = [I(I+1) - m(m+1)]^{1/2}|I, m+1\rangle, \tag{2-14}$$

and has matrix elements

$$\langle I, m'|\boldsymbol{I}_+|I, m\rangle = [I(I+1) - m(m+1)]^{1/2}\delta_{m+1, m'}. \tag{2-15}$$

The matrix representation of $\boldsymbol{I}_+$ for a single spin i**,** $\boldsymbol{I}_{i+}$**,** is efficiently expressed in the dimension of the Hilbert space of the spin itself, spanning $2(I_i+1)$. Examples are shown below for a spin 1/2 and a spin 1 particle

$$\boldsymbol{I}_{i+}\left(I_i = \frac{1}{2}\right) = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \qquad \boldsymbol{I}_{i+}(I_i = 1) = \sqrt{2}\begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix} \qquad \boldsymbol{I}_{i+}\left(I_i = \frac{3}{2}\right) = \begin{bmatrix} 0 & \sqrt{3} & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & \sqrt{3} \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Other useful relationships involving this operator are the following.

$$\boldsymbol{I}_{i+} = (\boldsymbol{I}_{ix} + i\boldsymbol{I}_{iy}) \qquad [\boldsymbol{I}_{i+}, \boldsymbol{I}_{iz}] = -\boldsymbol{I}_{i+} \qquad [\boldsymbol{I}_{i+}, \boldsymbol{I}_{i-}] = 2\boldsymbol{I}_{iz} \qquad \boldsymbol{I}_{i+} = (\boldsymbol{I}_{i-})^{\dagger} \tag{2-16}$$

and for the spin 1/2 case,

$$\boldsymbol{I}_{i+}\boldsymbol{I}_{i-} = \frac{1}{2} + \boldsymbol{I}_{iz} \qquad \boldsymbol{I}_{i-}\boldsymbol{I}_{i+} = \frac{1}{2} - \boldsymbol{I}_{iz} \tag{2-17}$$

It is trivial to verify the properties of the raising operator matrices in equation (2-14). When I=1/2 and I=1 we have

$$\boldsymbol{I}_+|\alpha\rangle = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}\begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} = 0 \qquad \boldsymbol{I}_+|\beta\rangle = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}\begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} = |\alpha\rangle$$

$$\boldsymbol{I}_+|\alpha\rangle = \sqrt{2}\begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} = 0 \qquad \boldsymbol{I}_+|\beta\rangle = \sqrt{2}\begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}\begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} = \sqrt{2}\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} = \sqrt{2}|\alpha\rangle \qquad \boldsymbol{I}_+|\gamma\rangle = \sqrt{2}\begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}\begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} = \sqrt{2}\begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} = \sqrt{2}|\beta\rangle$$

[1] Schiff, pg. 202, equation (27.25).

## 2.4.5 Single Spin I- Operator

The lowering operator $\boldsymbol{I}_-$ is defined to work on state $|I, m\rangle$ as[1]

$$\boldsymbol{I}_-|I, m\rangle = [I(I+1) - m(m-1)]^{1/2}|I, m-1\rangle \quad , \tag{2-18}$$

having matrix elements

$$\langle I, m'|\boldsymbol{I}_-|I, m\rangle = [I(I+1) - m(m-1)]^{1/2}\delta_{m-1, m'} \quad . \tag{2-19}$$

The matrix representation of $\boldsymbol{I}_-$ for a single spin i**,** $\boldsymbol{I}_{i-}$**,** is efficiently expressed in the dimension of the Hilbert space of the spin itself, spanning $2(I_i+1)$. Examples are shown below for particles with spin angular momentum 1/2, 1, and 3/2 respectively.

$$\boldsymbol{I}_{i-}\left(I_i = \frac{1}{2}\right) = \begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix} \qquad \boldsymbol{I}_{i-}(I_i = 1) = \sqrt{2}\begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \qquad \boldsymbol{I}_{i-}\left(I_i = \frac{3}{2}\right) = \begin{bmatrix} 0 & 0 & 0 & 0 \\ \sqrt{3} & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & \sqrt{3} & 0 \end{bmatrix} \tag{2-20}$$

Other useful relationships involving this operator are the following.

$$\boldsymbol{I}_{i-} = (\boldsymbol{I}_{ix} - i\boldsymbol{I}_{iy}) \qquad \boldsymbol{I}_{i-} = (\boldsymbol{I}_{i+})^{\dagger} \qquad [\boldsymbol{I}_{i-}, \boldsymbol{I}_{iz}] = \boldsymbol{I}_{i-} \qquad [\boldsymbol{I}_{i+}, \boldsymbol{I}_{i-}] = 2\boldsymbol{I}_{iz} \tag{2-21}$$

and in the spin 1/2 case,

$$\boldsymbol{I}_{i+}\boldsymbol{I}_{i-} = \frac{1}{2} + \boldsymbol{I}_{iz} \qquad \boldsymbol{I}_{i-}\boldsymbol{I}_{i+} = \frac{1}{2} - \boldsymbol{I}_{iz} \tag{2-22}$$

It is simple verify the properties of the lowering operator matrices in equation . For I=1/2 and I=1 we have

$$\boldsymbol{I}_-|\alpha\rangle = \begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix}\begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} = |\beta\rangle \qquad \boldsymbol{I}_-|\beta\rangle = \begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix}\begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} = 0$$

$$\boldsymbol{I}_-|\alpha\rangle = \sqrt{2}\begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} = \sqrt{2}\begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} = \sqrt{2}|\beta\rangle \qquad \boldsymbol{I}_-|\beta\rangle = \sqrt{2}\begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}\begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} = \sqrt{2}\begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} = \sqrt{2}|\gamma\rangle \qquad \boldsymbol{I}_-|\gamma\rangle = \sqrt{2}\begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}\begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} = 0$$

[1] Schiff, pg. 202, equation (27.25).

## 2.4.6 Spin Rotation Operators

### 2.4.6.1 *Definition*

A general rotation of angular momentum J of angle β about an axis u is defined as

$$\boldsymbol{R}_u(\beta) = exp[-i\beta(\boldsymbol{u} \bullet \boldsymbol{J})] = exp(-i\beta \boldsymbol{J}_u) \tag{2-23}$$

where $\boldsymbol{J_u}$ is the component of angular momentum along the axis u, and **u** a unit vector along the axis u.

### 2.4.6.2 *Single Spin I=1/2*

For a single spin, i, the rotation of spin angular momentum is then

$$\boldsymbol{R}_{iu}(\beta) = exp(-i\beta \boldsymbol{I}_{iu}) . \tag{2-24}$$

Explicit formulae for rotations of spin 1/2 species about any arbitrary axis is obtainable through use of the relationship between the single spin operators and the Pauli spin matrices below[1].

$$\sigma_x = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \qquad \sigma_y = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix} \qquad \sigma_z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \tag{2-25}$$

Direct comparison with equations ***, ***, and *** to the previous equation demonstrates that[2] for I = 1/2

$$\boldsymbol{I}_{iu}^{(1/2)} = \frac{1}{2}\sigma_u \text{ where } u \in \{x, y, z\} . \tag{2-26}$$

In fact, equation (2-26) is valid for any axis ***u***, not just the coordinate axes x, y and z. This is shown by expanding the general axis case in terms of the x, y, and z components.

$$\boldsymbol{I}_{iu}^{(1/2)} = u_x \boldsymbol{I}_{ix}^{(1/2)} + u_y \boldsymbol{I}_{iy}^{(1/2)} + u_z \boldsymbol{I}_{iz}^{(1/2)} = \frac{1}{2}[u_x\sigma_x + u_y\sigma_y + u_z\sigma_z] = \frac{1}{2}\sigma_u \tag{2-27}$$

In these equations, $\boldsymbol{u_x}$, $\boldsymbol{u_y}$, and $\boldsymbol{u_z}$ are the components of the *unit vector* in the direction of axis $u$. An alternative form of (2-26) is

$$\boldsymbol{I}_{iu}^{(1/2)} = \frac{1}{2}(\boldsymbol{u} \bullet \sigma) \tag{2-28}$$

[1] See Blum, pg. 4, Eq. (1.1.6), Schiff pg. 205, Eq. (27.30), and/or Gottfried, pg 275, Eq. (12).

[2] Note that the matrix forms of $\boldsymbol{I}_{ix}$, $\boldsymbol{I}_{iy}$, and $\boldsymbol{I}_{iz}$ used for comparison are in the single spin Hilbert space and independent of the index i.

It is easily verified from *** and (2-34) that the Pauli matrices are idempotent, that is, they have the property

$$\sigma_u^2 = 1 . \tag{2-29}$$

In turn, the single spin (I = 1/2) operators have the properties

$$[\boldsymbol{I}_{iu}^{(1/2)}]^{2m} = [1/2]^{2m}1 \tag{2-30}$$

and

$$[\boldsymbol{I}_{iu}^{(1/2)}]^{2m+1} = (1/2)^{2m}\boldsymbol{I}_{iu}^{(1/2)} = 2[1/2]^{2m+1}\boldsymbol{I}_{iu}^{(1/2)} \quad , \tag{2-31}$$

where m is an integer[1]. We now turn our attention back to the derivation of formulae for the rotations of spin 1/2 species about any arbitrary axis. From equation (2-24), we desire

$$\boldsymbol{R}_{iu}^{(1/2)}(\beta) = exp[-i\beta\boldsymbol{I}_{iu}^{(1/2)}] = \cos[\beta\boldsymbol{I}_{iu}^{(1/2)}] - i\sin[\beta\boldsymbol{I}_{iu}^{(1/2)}] \quad . \tag{2-32}$$

Applying the series expansions of sine and cosine,

$$\boldsymbol{R}_{iu}^{(1/2)}(\beta) = \sum_m (-1)^m \frac{[\beta\boldsymbol{I}_{iu}^{(1/2)}]^{2m}}{(2m)!} - i\sum_m (-1)^m \frac{[\beta\boldsymbol{I}_{iu}^{(1/2)}]^{2m+1}}{(2m+1)!} , \tag{2-33}$$

followed by the use of equations (2-30) and (2-31),

$$\boldsymbol{R}_{iu}^{(1/2)}(\beta) = 1\sum_m (-1)^m \frac{(\beta/2)^{2m}}{(2m)!} - i2\boldsymbol{I}_{iu}^{(1/2)}\sum_m (-1)^m \frac{(\beta/2)^{2m+1}}{(2m+1)!} , \tag{2-34}$$

produces a more compact formula,

$$\boldsymbol{R}_{iu}^{(1/2)}(\beta) = 1\cos(\beta/2) - i2\boldsymbol{I}_{iu}^{(1/2)}\sin(\beta/2) = 1\cos(\beta/2) - i\sigma_u\sin(\beta/2) . \tag{2-35}$$

We made use of equation *** in the previous step. Explicitly, we have

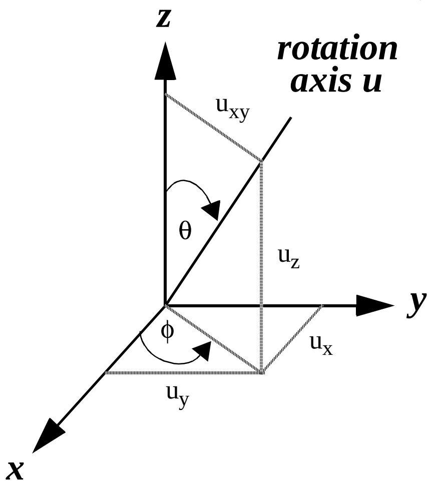
$$\boldsymbol{R}_{iu}^{(1/2)}(\beta) = \cos(\beta/2)\begin{bmatrix}1 & 0\\0 & 1\end{bmatrix} - i\sin(\beta/2)\left\{u_x\begin{bmatrix}0 & 1\\1 & 0\end{bmatrix} + u_y\begin{bmatrix}0 & -i\\i & 0\end{bmatrix} + u_z\begin{bmatrix}1 & 0\\0 & -1\end{bmatrix}\right\} \tag{2-36}$$

[1] Note that there is a mismatch of “powers to the m” between the left and right hand sides of equation (2-31). This must be so, as seen from the m=0 case, and results in the factor of 2 on the sine term of (2-35).

The general formula for a spin 1/2 particle is then[1]

$$\boldsymbol{R}_{iu}^{(1/2)}(\beta) = \begin{bmatrix} \cos\frac{\beta}{2} - iu_z\sin\frac{\beta}{2} & (-iu_x - u_y)\sin\frac{\beta}{2} \\ (-iu_x + u_y)\sin\frac{\beta}{2} & \cos\frac{\beta}{2} + iu_z\sin\frac{\beta}{2} \end{bmatrix}. \tag{2-37}$$

where $\boldsymbol{u_x}$, $\boldsymbol{u_y}$, and $\boldsymbol{u_z}$ are the components of the unit vector pointing in the direction of **u** and β is the angle of rotation. The following diagram shows these components with respect to the standard spherical coordinate system.



$$u_z = \cos\theta$$
$$u_{xy} = \sin\theta$$
$$u_x = u_{xy}\cos\phi = \sin\theta\cos\phi$$
$$u_y = u_{xy}\sin\phi = \sin\theta\sin\phi$$

Thus, equation (2-37) can be recast as (using $C\beta = \cos(\beta/2)$ $\quad S\beta = \sin(\beta/2)$)

$$\begin{aligned} \boldsymbol{R}_{iu}^{(1/2)}(\beta) &= \begin{bmatrix} C\beta - i\cos\theta S\beta & (-i\cos\phi - \sin\phi)\sin\theta S\beta \\ (-i\cos\phi + \sin\phi)\sin\theta S\beta & C\beta + i\cos\theta S\beta \end{bmatrix} \\ &= \begin{bmatrix} C\beta - i\cos\theta S\beta & -ie^{-i\phi}\sin\theta S\beta \\ -ie^{i\phi}\sin\theta S\beta & C\beta + i\cos\theta S\beta \end{bmatrix} \end{aligned} \tag{2-38}$$

## 2.4.6.3 *Single Spin I>1/2*

The rotation formulas involving $\boldsymbol{R}_{iu}^{(1/2)}$ are exclusively applicable to I=1/2 spins because equation (2-26) is not valid for I>1/2. In cases involving spins with I > 1/2 one is forced to use the general formula, equation (2-24), which is reproduced below.

$$\boldsymbol{R}_{iu}(\beta) = exp(-i\beta\boldsymbol{I}_{iu}) . \tag{2-39}$$

[1] See Cohen-Tannoudji, Volume Two, pages 983-985, Sakurai, pages 165-166, or Goldman page 64.

Not only is this more difficult to directly evaluate, it is computationally more expensive to perform. Fortunately, one may usually work with $\boldsymbol{R}_{in}(\beta)$ in the Hilbert space of the spin itself and, for example in the case of a spin with **I**=1 will only be a (3x3) matrix. Expanding $\boldsymbol{I}_{iu}$ in the previous formula with the equation analogous to (2-27) without regard to the spin quantum number,

$$\boldsymbol{I}_{iu} = u_x\boldsymbol{I}_{ix} + u_y\boldsymbol{I}_{iy} + u_z\boldsymbol{I}_{iz} \tag{2-40}$$

produces

$$\boldsymbol{R}_{iu}(\beta) = exp(-i\beta\boldsymbol{I}_{iu}) = exp[-i\beta(u_x\boldsymbol{I}_{ix} + u_y\boldsymbol{I}_{iy} + u_z\boldsymbol{I}_{iz})] \tag{2-41}$$

From the previous trigonometric relationships of  this is equivalent to

$$\boldsymbol{R}_{iu}(\beta) = exp\{-i\beta[(\sin\theta\cos\phi)\boldsymbol{I}_{ix} + (\sin\theta\sin\phi)\boldsymbol{I}_{iy} + \cos\theta\boldsymbol{I}_{iz}]\} \tag{2-42}$$

$$\boldsymbol{R}_{iu}(\beta) = e^{(-i\beta\sin\theta\cos\phi\boldsymbol{I}_{ix})}e^{(-i\beta\sin\theta\sin\phi\boldsymbol{I}_{iy})}e^{(-i\beta\cos\theta\boldsymbol{I}_{iz})} \tag{2-43}$$

Still under construction ********* SOSI (2-44)

$$\boldsymbol{R}_{iu}(\beta) = \boldsymbol{R}_{iz}(\phi)\boldsymbol{R}_{iy}(-\theta)\boldsymbol{R}_{iz}(\beta)[\boldsymbol{R}_{iy}(-\theta)]^{-1}[\boldsymbol{R}_{iz}(\phi)]^{-1} \tag{2-45}$$

## 2.4.6.4 Rotation Operators About X

The formulas used to determine individual $\boldsymbol{R}_{ix}(\beta)$ matrices are obtained from equation (2-35) in the case of a spin 1/2 particle,

$$\boldsymbol{R}_{ix}^{(1/2)}(\beta) = 1\cos(\beta/2) - i2\boldsymbol{I}_{ix}\sin(\beta/2) \quad , \tag{2-46}$$

and from the general equation (2-24) in the case of a spin with I > 1/2,

$$\boldsymbol{R}_{ix}(\beta) = exp(-i\beta\boldsymbol{I}_{ix}) \ . \tag{2-47}$$

Substitution of equation *** into equation *** yields the explicit rotation matrix for this single spin 1/2 treatment.[1]

$$\boldsymbol{R}_{ix}^{(1/2)}(\beta) = \begin{bmatrix} \cos(\beta/2) & -i\sin(\beta/2) \\ -i\sin(\beta/2) & \cos(\beta/2) \end{bmatrix} \tag{2-48}$$

Unfortunately, there is currently no similar simple formula for a spin with I > 1/2 and these are computed by actual exponentiation of the operator as stated in ***. A few specific rotation matrices one will obtain from this function

[1] See Ernst, Bodenhausen, and Wokaun, page 405, equation (8.1.4).

are as follows, where ***a*** = 2.45 and ***b*** = sqrt(2).

$$\boldsymbol{R}_{ix}^{(1/2)}\left(\frac{\pi}{3}\right) = \begin{bmatrix} 0.866 & -0.5i \\ -0.5i & 0.866 \end{bmatrix} \qquad \boldsymbol{R}_{ix}^{(1/2)}\left(\frac{\pi}{2}\right) = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 & -i \\ -i & 1 \end{bmatrix} \qquad \boldsymbol{R}_{ix}^{(1/2)}(\pi) = \begin{bmatrix} 0 & -i \\ -i & 0 \end{bmatrix}$$

$$\boldsymbol{R}_{ix}^{(1)}\left(\frac{\pi}{3}\right) = \frac{1}{4}\begin{bmatrix} 3 & -ai & -1 \\ -ai & 2 & -ai \\ -1 & -ai & 3 \end{bmatrix} \qquad \boldsymbol{R}_{ix}^{(1)}\left(\frac{\pi}{2}\right) = \frac{1}{2}\begin{bmatrix} 1 & -bi & -1 \\ -bi & 0 & -bi \\ -1 & -bi & 1 \end{bmatrix} \qquad \boldsymbol{R}_{ix}^{(1)}(\pi) = \frac{-1}{2}\begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix}$$

$$\boldsymbol{R}_{ix}^{(3/2)}\left(\frac{\pi}{2}\right) = \begin{bmatrix} 0.35 & -0.61i & -0.61 & 0.35i \\ -0.61i & -0.35 & -0.35i & -0.61 \\ -0.61 & -0.35i & -0.35 & -0.61i \\ 0.35i & -0.61 & -0.61i & 0.35 \end{bmatrix} \qquad \boldsymbol{R}_{ix}^{(3/2)}(\pi) = \begin{bmatrix} 0 & 0 & 0 & i \\ 0 & 0 & i & 0 \\ 0 & i & 0 & 0 \\ i & 0 & 0 & 0 \end{bmatrix}$$

For a multiple spin system one may obtain the rotation matrix in the composite Hilbert space by taking direct products of the single spin operators in accordance with equation (2-25).

Finally, we consider the effect of these rotations on the state vectors of a spin 1/2 particle. It we use

$$|\alpha\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \qquad |\beta\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \tag{2-49}$$

Then a rotation by β degrees about x has the following effect.

$$\begin{aligned} [\boldsymbol{R}_{x}^{(1/2)}(\beta)]|\alpha\rangle &= \begin{bmatrix} \cos(\beta/2) & -i\sin(\beta/2) \\ -i\sin(\beta/2) & \cos(\beta/2) \end{bmatrix}\begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} \cos(\beta/2) \\ -i\sin(\beta/2) \end{bmatrix} \\ [\boldsymbol{R}_{x}^{(1/2)}(\beta)]|\beta\rangle &= \begin{bmatrix} \cos(\beta/2) & -i\sin(\beta/2) \\ -i\sin(\beta/2) & \cos(\beta/2) \end{bmatrix}\begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} -i\sin(\beta/2) \\ \cos(\beta/2) \end{bmatrix} \end{aligned} \tag{2-50}$$

Then a rotation by β degrees about x has the following effect.

$$\begin{aligned} \left[\boldsymbol{R}_{x}^{(1/2)}\left(\frac{\pi}{2}\right)\right]|\alpha\rangle &= \frac{1}{\sqrt{2}}\begin{bmatrix} 1 \\ -i \end{bmatrix} \qquad [\boldsymbol{R}_{x}^{(1/2)}(\pi)]|\alpha\rangle = \begin{bmatrix} 0 \\ -i \end{bmatrix} \\ \left[\boldsymbol{R}_{x}^{(1/2)}\left(\frac{\pi}{2}\right)\right]|\beta\rangle &= \frac{1}{\sqrt{2}}\begin{bmatrix} -i \\ 1 \end{bmatrix} \qquad [\boldsymbol{R}_{x}^{(1/2)}(\pi)]|\beta\rangle = \begin{bmatrix} -i \\ 0 \end{bmatrix} \end{aligned} \tag{2-51}$$

Substitution of equation *** into equation *** yields the explicit rotation matrix for this single spin 1/2 treatment.

$$\boldsymbol{R}_{y}(\beta) = \begin{bmatrix} \cos(\beta/2) & -\sin(\beta/2) \\ \sin(\beta/2) & \cos(\beta/2) \end{bmatrix}. \tag{2-52}$$

Unfortunately, there is currently no similar simple formula for a spin with I > 1/2 and these are computed by actual exponentiation of the operator as stated in ***. A few specific rotation matrices one will obtain from this function are as follows.

$$\boldsymbol{R}_{iy}^{(1/2)}\left(\frac{\pi}{3}\right) = \begin{bmatrix} 0.866 & -0.5 \\ 0.5 & 0.866 \end{bmatrix} \qquad \boldsymbol{R}_{iy}^{(1/2)}\left(\frac{\pi}{2}\right) = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 & -1 \\ 1 & 1 \end{bmatrix} \qquad \boldsymbol{R}_{iy}^{(1/2)}(\pi) = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \tag{2-53}$$

One may obtain the rotation matrix in a multiple spin system by taking direct products of the single spin operators in accordance with equation (2-3). This particular rotation corresponds directly to the reduced Wigner rotation matrices[1]

$$\boldsymbol{R}_{iy}^{(1/2)}(\beta) = d^{(1/2)}(\beta) \quad . \tag{2-54}$$

## 2.4.6.5 Rotation Operators About Z

Substitution of equation *** into equation *** will yield the explicit rotation matrix for this single spin 1/2 treatment.[2] Similarly, by blending equation *** with equation *** one may obtain the matrix representation of $\mathbf{R}_z(\beta)$ for a single spin 1 particle. These are shown in the following equation.

$$\boldsymbol{R}_z^{\frac{1}{2}}(\beta) = \begin{bmatrix} exp[-i(\beta/2)] & 0 \\ 0 & exp[i(\beta/2)] \end{bmatrix} \qquad \boldsymbol{R}_z^{1}(\beta) = \begin{bmatrix} exp(-i\beta) & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & exp(i\beta) \end{bmatrix}, \tag{2-55}$$

Unfortunately, there is currently no simple general formula for a spin with arbitrary I and these are computed by actual exponentiation of the operator as stated in ***. A few specific rotation matrices one will obtain from this function are as follows.

$$\boldsymbol{R}_{iz}^{(1/2)}\left(\frac{\pi}{3}\right) = \begin{bmatrix} a* & 0 \\ 0 & a \end{bmatrix} \qquad \boldsymbol{R}_{iz}^{(1/2)}\left(\frac{\pi}{2}\right) = \frac{1}{\sqrt{2}}\begin{bmatrix} 1+i & 0 \\ 0 & 1+i \end{bmatrix} \qquad \boldsymbol{R}_{iz}^{(1/2)}(\pi) = \begin{bmatrix} -i & 0 \\ 0 & i \end{bmatrix}$$

$$a = 0.866 + 0.5i$$

One may obtain the rotation matrix in a multiple spin system by taking direct products of the single spin operators in accordance with equation (2-1).

[1] See Brink and Satchler, page 25.

[2] See Goldman, page 59, equation (3.36).

# 3 Base Spin System

## 3.1 Overview

The class SpinSys (***spin_sys***) defines the basic physical attributes of a system of spins. These essential quantities are the number of spins and their associated spin angular momentum. Functions are provided for simplified access to all spin sys quantities and for disk I/O.

## 3.2 Available Functions

### Algebraic Operators

| | | |
|---|---|---|
| spin_sys | - Constructor | page 28 |
| = | - Assignment | page 28 |
| == | - Equality | page 29 |
| != | - Inequality | page 29 |

### Basic Functions

| | | |
|---|---|---|
| spins | - Number of spins | page 31 |
| size | - Number of spins (same as spins) | page 31 |
| spinpairs | - Number of spin pairs | page 31 |
| HS | - Retrieve spin or spin system Hilbert space | page 32 |
| isotope | - Set or retrieve spin isotope type | page 33 |
| symbol | - Spin isotope type as a string, e.g. 19F. | page 33 |
| qn | - Spin angular momentum of  spin or system. | page 34 |
| element | - Spin element type as a string, e.g. Carbon. | page 34 |
| momentum | - Spin angular momentum as a string. | page 35 |
| gamma | - Gyromagnetic ratio of a spin. | page 36 |
| qState | - State vector of a particular quantum state | page 36 |
| qStates | - Matrix describing all quantum states | page 37 |
| qnState | - Get the quantum number of a state | page 38 |
| qnStates | - Get the quantum number of all states | page 39 |
| qnDist | - Get a statistic over the quantum numbers | page 40 |
| CoherDist | - Get a statistic over the different coherences | page 40 |
| homonuclear | - Test whether system is homonuclear. | page 41 |
| heteronuclear | - Test whether system is heteronuclear. | page 41 |
| spinhalf | - Test whether system has all I=1/2 spins. | page 42 |
| electrons | - Test whether system has any electrons. | page 42 |
| nepair | - Test whether spin pair is electron-nucleon. | page 42 |
| pairidx | - Integer index of spin pairing. | page 43 |
| isotopes | - Retrieve isotope count or isotope symbol | page 44 |

### Spin Flags Functions

| | | |
|---|---|---|
| SetFlag | - Set a spin flag true/false status. | page 45 |
| SetFlags | - Set selected spin flags true/false status. | page 45 |
| GetFlag | - Get a spin flag true/false status. | page 46 |
| GetFlags | - Get vector of spin flags. | page 46 |
| name | - Set or retrieve spin system name. | page 47 |

## Input/Output

| | | |
|---|---|---|
| = | - Assignment of a spin sys to/from a p_set | page 48 |
| += | - Addition of a spin sys to a p_set | page 48 |
| write | - Write spin sys to disk file (as a parameter set). | page 49 |
| read | - Read spin sys from disk file (from parameter set). | page 49 |
| print | - Send spin system to output stream. | page 50 |
| << | - Send spin system to an output stream | page 50 |

# 3.3 Algebraic Operators

## 3.3.1 spin_sys

### Usage:

#include <HSLib/SpinSys.h>
void spin_sys::spin_sys()
void spin_sys::spin_sys(int nspins)
void spin_sys::spin_sys(const spin_sys &sys)

### Description:

The function *spin_sys* is used to create a new basic spin system.

1. ***spin_sys()*** - Called without arguments the function creates an "empty" NULL spin system. A spin system constructed with this command will not have full spin system capabilities until either the number of spins has been assigned or the system itself has been set equal to another spin system.
2. ***spin_sys(int nspins)*** - Called with an integer, the function will create a spin system of the size indicated. By default, all spins are set to be protons (spin I = 1/2). All other spin system parameters are left unassigned although the appropriate array space for storage of all spin system parameters will be allocated
3. ***spin_sys(spin_sys &sys)*** - Called with another spin system ***sys***, the function will make a new spin system which is a copy ***sys***.

### Return Value:

spin_sys returns no parameters. It is used strictly to create a spin_sys.

### Examples:

```
#include <gamma.h>
main()
  {
  spin_sys A;                          // Define all NULL spin system called A.
  spin_sys A2BX(4);                    // Define spin system A2BX containing four spins.
  spin_sys Four(A2BX);                 // Define spin system Four identical to current A2BX.
  A = Four;                            // Also a valid way to set A to be equal to Four.
  A.read("ABX.sys");                   // A now read in from disk file "ABX.sys".
  }
```

**See Also: =, read**

## 3.3.2 =

### Usage:

#include <HSLib/SpinSys.h>
void spin_sys::operator = (spin_sys &sys)

### Description:

The unary ***operator =*** (the assignment operator) is allows for the setting of one spin_sys to another spin_sys. If the spin system exists it will be overwritten by the assigned spin_sys.

**Return Value:**

None, the function is void

**Examples:**

```
#include <gamma.h>
main()
  {
  spin_sys A;                              // Define all NULL spin system called A.
  spin_sys A2BX(4);                        // Define spin system A2BX containing four spins.
  A = A2BX;                                // Set spin system A to be equal to current Four.
  }
```

**See Also: spin_sys, read**

## 3.3.3 ==

**Usage:**

#include <HSLib/SpinSys.h>
int spin_sys::operator == (spin_sys &sys)

**Description:**

The ***unary operator* ==** (the equality operator) will test two spin systems for their equality. If the two spin systems are identical the function returns true. If the two systems are not equal the function returns false. This function does not check the settings of the spin flags!

**Return Value:**

The function returns an integer which equates to TRUE or FALSE.

**Example:**

```
#include <gamma.h>
main()
  {
  spin_sys AX;                             // Define a NULL spin system called AX.
  spin_sys A2BX(4);                        // Define spin system A2BX containing four spins.
  AX = A2BX;                               // Set AX equal to A2BX.
  if(A == A2BX)                            // See if they are in fact the same.
    cout << “The two spin systems are identical”;
  }
```

**See Also: !=**

## 3.3.4 !=

**Usage:**

#include <HSLib/SpinSys.h>
int spin_sys::operator != (spin_sys &sys)

**Description:**

The ***unary operator* !=** (the inequality operator) can be used to test whether two spins are equal. The function returns true if the two systems being compared are not equal and false if they are identical. This function does not check the settings of the spin flags!

## Return Value:

The function returns an integer which equates to TRUE or FALSE.

## Example:

```
#include <gamma.h>
main()
  {
  spin_sys AX(2);                          // Define a two spin system called AX.
  spin_sys A2BX(4);                        // Define spin system A2BX containing four spins.
  if(AX != A2BX)                           // See if they are not the same.
    cout << “The two spin systems are different”;
  }
```

## See Also: ==

## 3.4 Basic Functions

### 3.4.1 spins

### 3.4.2 size

**Usage:**

#include <HSLib/SpinSys.h>
int spin_sys::spins( )
int spin_sys::size( )

**Description:**

The function ***spins*** returns the number of spins in the system as an integer value. Information regarding the spin system itself cannot be altered by this function.[1]

**Return Value:**

***spins*** returns an integer value that is the number of spins in the spin system.

**Example:**

```
#include <gamma.h>
main()
  {
  int i;
  spin_sys xyz(5);                    // Define spin system xyz containing five spins.
  i = xyz.spins();                    // Retrieve the size (5) of spin system xyz.
  cout << xyz.spins();                // Print the size of spin system xyz (5).
  }
```

**See Also:**

### 3.4.3 spinpairs

**Usage:**

#include <HSLib/SpinSys.h>
int spin_sys::spinpairs ( )

**Description:**

***spinpairs*** will count the number of spin pairs in the system.

**Return Value:**

Integer value for the number of spin pairs.

**Example:**

[1] The function “size” is equivalent to the function “spins”. It is present because it has a more intuitive function name but can conflict with other functions named “size” in classes derived from spin_sys. For all GAMMA uses size will work as well a spins for any spin_sys. For derived classes, spins is preferable for it avoids type casting. Function size will work for derived classes but may need a type cast as spin_sys.

```
#include <gamma.h>
main()
  {
  int i;
  spin_sys CH3(3);                          // Define spin system CH3 containing three spins.
  cout << CH3.spinpairs();                  // Output number of spin pairs (3 in this case).
  }
```

**See Also:**

## 3.4.4 HS

**Usage:**

```
#include <HSLib/SpinSys.h>
int spin_sys::HS ( )
int spin_sys::HS (int spin)
```

**Description:**

***HS*** is used to access the dimension of the Hilbert space associated with either the entire spin system or an individual spin. The spin system Hilbert space (or composite Hilbert space) is the product of the individual spin Hilbert spaces, formally given by equation (3-2) on page 51. For a single spin, the spin Hilbert space size is related to the spin quantum number by equation equation (3-1) on page 51. These are given by

$$HS = \prod_{i=1}^{nspins} HS(i) = \prod_{i=1}^{nspins} (2I_i + 1) .$$

where ***i*** is the spin label and $\boldsymbol{I_i}$ the associated spin quantum number.

**Return Value:**

Integer value for the dimension of the Hilbert space.

**Example:**

```
#include <gamma.h>
main()
  {
  int i;
  spin_sys CH3(3);                          // Define spin system CH3 containing three spins.
  cout << CH3.HS();                         // Output Hilbert space size (8 by default, all I=1/2);
  cout << CH3.HS(1);                        // Output 2nd spin Hilbert space size (2 since its 1H)
  }
```

**See Also:**

## 3.4.5 isotope

### Usage:

#include <HSLib/SpinSys.h>

void spin_sys::isotope (int, const string &)

void spin_sys::isotope (int, const Isotope &)

Isotope spin_sys::isotope (int) const

### Description

The function ***isotope*** is used to assign an isotope label to a spin. This can be done by specifying which spin is to be labeled and the specifying which isotope it is to be labeled with. The isotope can be given in the form of a string such as “13C” or “2H”. Alternatively the isotope can be specified directly (see class Isotope).

### Return Value:

The function is overloaded to either set or retrieve a particular spin isotope. Used to set the isotope type, the function is void and returns nothing. Used to retrieve an isotope type the function returns the isotope.

### Example(s):

```
#include <gamma.h>
main()
  {
  spin_sys xyz(5);                          // Define spin system xyz containing five spins.
  xyz.isotope(0, ”3H”);                     // Set isotope of spin 0 in spin system xyz to be 3H.
  cout << “Spin 0: “ << xyz.isotope(0);     // Output isotope tye of spin 0.
  }
```

**See Also: symbol**

## 3.4.6 symbol

### Usage:

#include <HSLib/SpinSys.h>

string spin_sys::symbol(int) const

### Description:

*symbol* is used to obtain, in string format, the isotope type of a spin as set by the function *isotope*. It does not alter the spin system and is set to a default value of 1H. This function is typically used for formatted output.

### Return Value:

None.

### Example(s):

```
#include <gamma.h>
main()
  {
  CH3.spin_sys(3);                          // define spin system CH3 containing three spins.
  CH3.isotope(2,”23Na”);
  cout << CH3.symbol(2);                     // Prints isotope type 23Na of spin 2 in CH3.
  }
```

**See Also: isotope, element**

## 3.4.7 qn

**Usage:**

```
#include <HSLib/SpinSys.h>
double spin_sys::qn (int) const
double spin_sys::qn () const
```

**Description:**

*qn* is used to obtain the quantum number I (spin angular momentum) carried by a spin. These are output in units of hbar, the default (for proton) being 0.5. When no spin is specified the function returns the sum of all spin I's.

**Return Value:**

A floating point number, double precision.

**Example(s):**

```
#include <gamma.h>
main()
  {
  CH3.spin_sys(3);                      // define spin system CH3 containing three spins.
  CH3.isotope(2,"23Na");
  cout << CH3.qn (2);                   // Prints I value 1.5 of spin 2 in spin system CH3.
  }
```

**See Also: momentum, isotope**

## 3.4.8 element

**Usage:**

```
#include <HSLib/SpinSys.h>
string spin_sys::element (int) const
```

**Description:**

*element* is used to obtain the name of the element assigned to a spin via the function *isotope*. It does not alter the spin system and will return a blank if the spin has not previously been assigned. The function is commonly used for formatted output.

**Return Value: Pointer to a string containing element label.**

**Example(s):**

```
#include <gamma.h>
main()
  {
  CH3.spin_sys(3);                      // define spin system CH3 containing three spins.
  isotope(0, "23Na");
  cout << CH3.element(0);                // prints element label Sodium, spin 0, system CH3.
  }
```

**See Also: isotope, symbol.**

## 3.4.9 momentum

### Usage:

#include <HSLib/SpinSys.h>
char* spin_sys::momentum (int) const

### Description:

*momentum* is used to obtain, in string format, the spin angular momentum carried by a spin as set by the function *isotope*. It does not alter the spin system and is set to a default value of 1/2. This function is used for formatted output.

### Return Value:

The function returns a pointer to a string.

### Example:

```
#include <gamma.h>
main()
  {
  CH3.spin_sys(3);                      // define spin system CH3 containing three spins.
  CH3.isotope(2,”23Na”);
  cout << CH3.momentum (2);             // Prints momentum 3/2 of spin 2 in spin system CH3.
  }
```

**See Also: qn, isotope**

## 3.4.10 gamma

### Usage:

#include <HSLib/SpinSys.h>
double spin_sys::gamma (int) const

### Description:

*gamma* will return a value for the gyromagnetic ratio of a spin as assigned by the function *isotope*. It does not alter the spin system and will default to the proton value if the spin has not previously had an isotope assignment. Gamma values are used to set relative frequencies and in some relaxation computations. In GAMMA, gyromagnetic ratios are maintained in SI units, $T^{-1}$ $sec^{-1}$. For example, the returned value for $^{19}F$ is 2.51719 x $10^8$ and that of a proton 2.67519 x $10^8$.

### Return Value:

Double, gyromagnetic ratio of spin with units rad $T^{-1}$ $sec^{-1}$.

### Example:

```
#include <gamma.h>
main()
  {
  CH3.spin_sys(3);                    // Define spin system CH3 containing three spins.
  cout << CH3.gamma(2);               // Prints gyromagnetic ratio spin 2, 2.67519e+08.
  }
```

**See Also: isotope.**

## 3.4.11 qState

### Usage:

#include <HSLib/SpinSys.h>
row_vector spin_sys::qState (int) const

### Description:

The function *qState*s will return a row vector whose elements correspond the spin quantum states for the particular spin of the particular basis function indicated. The vector elements are given mathematically by the formula

$$[\langle 1|v(i)|j\rangle]|\Phi_i\rangle = m_{zj}|\Phi_i\rangle = \hat{I}_{zj}|\Phi_i\rangle .$$

where ν***(i)*** is the vector returned by the function, $\boldsymbol{I_{zj}}$ is the spin operator along the z-axis for spin ***j***, and $|\Phi_i\rangle$ the $\boldsymbol{i^{th}}$ product basis function where ***i*** is specified as the function argument[1].

### Return Value:

A row vector is returned.

### Example:

[1] For a description of the default basis functions see the Chapter Class Spin Operator in the GAMMA User Documentation.

```
#include <gamma.h>
main()
  {
  spin_sys sys(5);                          //Create a 5 spin system, all I=1/2 by default
  cout sys.qState(31);                      //Write the vector for the last basis function
  }
```

**See Also: qStates**

## 3.4.12 qStates

### Usage:

#include <HSLib/SpinSys.h>
matrix spin_sys::qStates () const

### Description:

The function *qState*s will return a matrix having rows corresponding to the product basis functions associated with the spin system Hilbert space and having columns associated with the spins of the system. The elements of the matrix are the spin quantum states for the particular spin of the particular basis function. The matrix elements are mathematically given by the formula

$$[\langle i|mx|j\rangle]|\Phi_i\rangle = m_{zj}|\Phi_i\rangle = \hat{I}_{zj}|\Phi_i\rangle,$$

where ***mx*** is the matrix returned by the function, $\boldsymbol{I_{zj}}$ is the spin operator along the z-axis for spin ***j***, and $|\Phi_i\rangle$ the $\boldsymbol{i^{th}}$ product basis function[1]. The matrix a row dimension thus equals the number of product basis functions possible (the composite Hilbert space size of the spin system)and matrix column dimension is the number of spins in the system. This function easily understood from the following figure[2].

*Matrix Examples Returned by Function qStates*

*A.*

$$\begin{array}{c|cc|} & \mathbf{0} & \mathbf{1} \\ \alpha\alpha & 0.50 & 0.50 \\ \alpha\beta & 0.50 & -0.50 \\ \beta\alpha & -0.50 & 0.50 \\ \beta\beta & -0.50 & -0.50 \end{array}$$

*B.*

$$\begin{array}{c|cc|} & \mathbf{0} & \mathbf{1} \\ \alpha\alpha & 0.50 & 1 \\ \alpha\beta & 0.50 & 0 \\ \alpha\gamma & 0.50 & -1 \\ \beta\alpha & -0.50 & 1 \\ \beta\beta & -0.50 & 0 \\ \beta\gamma & -0.50 & -1 \end{array}$$

*C.*

$$\begin{array}{c|ccc|} & \mathbf{0} & \mathbf{1} & \mathbf{2} \\ \alpha\alpha\alpha & 0.50 & 0.50 & 0.50 \\ \alpha\alpha\beta & 0.50 & 0.50 & -0.50 \\ \alpha\beta\alpha & 0.50 & -0.50 & 0.50 \\ \alpha\beta\beta & 0.50 & -0.50 & -0.50 \\ \beta\beta\alpha & -0.50 & 0.50 & 0.50 \\ \beta\alpha\beta & -0.50 & 0.50 & -0.50 \\ \beta\beta\alpha & -0.50 & -0.50 & 0.50 \\ \beta\beta\beta & -0.50 & -0.50 & -0.50 \end{array}$$

*Figure 4-1* : Matrices returned from the function qStates. A - A two spin system both having I=1/2. B- A two spin system, the first with I=1/2 and the second with I=1. C - A three spin system, all I=1/2.

[1] For a description of the default basis functions see the Class Spin Operator Documentation.

[2] The figure arrays were generated by the example program qStates.cc included at the end of this Chapter.

**Return Value:**

A matrix is returned.

**Example:**

```
#include <gamma.h>
main()
  {
  spin_sys sys(3);                              // Create a 3 spin system, all I=1/2 by default
  cout sys.qStates();                           // Send the matrix to standard output
  }
```

**See Also: qstate**

## 3.4.13 qnState

**Usage:**

#include <HSLib/SpinSys.h>
double spin_sys::qnState (int) const

**Description:**

The function *qState* will return the spin quantum value of the indicated product basis function associated with the spin system Hilbert space. This is given mathematically by the formula

$$m_z|\Phi_i\rangle = \hat{I}_z|\Phi_i\rangle,$$

where value returned by the funciton is $\boldsymbol{m_z}$ , and $|\Phi_i\rangle$ is the requested product basis function. The interger ***i*** is supplied as a function argument.

**Return Value:**

A double is returned.

**Example:**

```
#include <gamma.h>
main()
  {
  spin_sys sys(3);                              //Create a 3 spin system, all I=1/2 by default
  cout sys.qnState(6);                          //Output I for the basis function 6
  }
```

**See Also: qnStates, qState, qStates**

## 3.4.14 qnStates

### Usage:

#include <HSLib/SpinSys.h>
col_vector spin_sys::qnStates () const

### Description:

*qnState*s will return a vector containing the quantum number of all state vectors associated with the spin system Hilbert space.

### ption:

The function *qnState*s will return a  row vector with a dimension equal to the spin system composite Hilbert space. The elements of the vector are the spin quantum numbers of the corresponding product basis functions. Th elements of the returned vector are given mathematically by the formula

$$[\langle 1|v|i\rangle]|\Phi_i\rangle = m_z|\Phi_i\rangle = \hat{I}_z|\Phi_i\rangle \quad .$$

where $v$ is the vector returned by the function, $\hat{I}_z$ is the total spin angular momentum operator along the z-axis, and $|\Phi_i\rangle$ the $i^{th}$ product basis function[1]. This function easily understood from the following figure.

*Vector Examples Returned by Function qnStates*

*A.*

$$\begin{array}{c} \alpha\alpha \\ \alpha\beta \\ \beta\alpha \\ \beta\beta \end{array} \begin{bmatrix} 1.00 \\ 0.00 \\ 0.00 \\ -1.00 \end{bmatrix}$$

*B.*

$$\begin{array}{c} \alpha\alpha \\ \alpha\beta \\ \alpha\gamma \\ \beta\alpha \\ \beta\beta \\ \beta\gamma \end{array} \begin{bmatrix} 0.50 \\ 0.50 \\ 0.50 \\ -0.50 \\ -0.50 \\ -0.50 \end{bmatrix}$$

*C.*

$$\begin{array}{c} \alpha\alpha\alpha \\ \alpha\alpha\beta \\ \alpha\beta\alpha \\ \alpha\beta\beta \\ \beta\alpha\alpha \\ \beta\alpha\beta \\ \beta\beta\alpha \\ \beta\beta\beta \end{array} \begin{bmatrix} 1.50 \\ 0.50 \\ 0.50 \\ -0.50 \\ 0.50 \\ -0.50 \\ -0.50 \\ -1.50 \end{bmatrix}$$

***Figure 4-2*** **: Vectors returned from the function qnStates. A - A two spin system both having I=1/2. B- A two spin system, the first with I=1/2 and the second with I=1. C - A three spin system, all I=1/2.**

### Return Value:

A column vector is returned.

### Example:

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

[1] For a description of the default basis functions see the Chapter Class Spin Operator in the GAMMA User Documentation.

```
    spin_sys sys(3);                              //Create a 3 spin system, all I=1/2 by default
    cout sys.qStates( );                          //Output the vector of basis function I'sf
    }
```

**See Also: qState, qStates, qnState**

## 3.4.15 qnDist

**Usage:**

```
#include <HSLib/SpinSys.h>
row_vector spin_sys::qnDist () const
```

**Description:**

*qnDist* will return a vector of the distribution of quantum states associated with the spin system Hilbert space.

**Return Value:**

A row vector is returned.

**Example(s):**

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

**See Also:**

## 3.4.16 CoherDist

**Usage:**

```
#include <HSLib/SpinSys.h>
row_vector spin_sys::CoherDist () const
```

**Description:**

*CoherDist* will return a vector of the distribution of quantum coherences associated with the spin system Hilbert space.

**Return Value:**

A row vector is returned.

**Example(s):**

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

## 3.4.17 homonuclear

**Usage:**

```
#include <HSLib/SpinSys.h>
int spin_sys::homonuclear() const
```

**Description:**

The function *homonuclear* is used to test whether a spin system is homonuclear. If it is homonuclear the function will return TRUE else it will return FALSE.

**Return Value:**

An integer.

**Example(s):**

```
#include <gamma.h>
main()
  {
  AX2.spin_sys(3);                        // define spin system AX2 containing three spins.
  if(AX2.homonuclear())
     cout << "\nHomonuclear System";      // AX2 contains all protons by default.
  AX2.isotope(0,"3H");                    // Set first spin to tritium
  if(!CH3.homonuclear())
     cout << "\nHeteronuclear System";    // AX2 is longer homonuclear.
  }
```

**See Also: heteronuclear**

## 3.4.18 heteronuclear

**Usage:**

```
#include <HSLib/SpinSys.h>
int spin_sys::heteronuclear( ) const
```

**Description:**

The function *heteronuclear* is used to test whether a spin system is heteroonuclear. If it is heteronuclear the function will return TRUE else it will return FALSE.

**Return Value:**

An integer.

**Example(s):**

```
#include <gamma.h>
main()
  {
  AX2.spin_sys(3);                        // define spin system AX2 containing three spins.
  if(AX2.heteronuclear())
     cout << "\nHeteronuclear System";    // This won't print cause AX2 all 1H by default.
  AX2.isotope(0,"3H");                    // Set first spin to tritium
  if(CH3.heteronuclear())
     cout << "\nHeteronuclear System";    // AX2 is now heteronuclear, this will print.
  }
```

**See Also: homonuclear**

## 3.4.19 spinhalf

### Usage:

```
#include <HSLib/SpinSys.h>
int spin_sys::spinhalf( ) const
```

### Description:

The function *spinhalf* is used to test whether a spin system is composed entirely of spin 1/2 particles. If so the function will return TRUE else it will return FALSE.

### Return Value:

An integer.

### Examples:

```
#include <gamma.h>
main()
  {
  AX2.spin_sys(3);                          // define spin system AX2 containing three spins.
  if(AX2.spinhalf())
     cout << “\nAll spins are I=1/2”;       // This will print cause  all 1H by default.
  AX2.isotope(0,”2H”);                      // Set first spin to deuterium
  if(CH3.spinhalf())
     cout << “\nNot all spins are I=1/2”;   // AX2 is now has an I=1 spin, this will print.
  }
```

**See Also: homonuclear**

## 3.4.20 electrons

### Usage:

```
#include <HSLib/SpinSys.h>
int spin_sys::electrons( ) const
```

### Description:

The function *electrons* is used to test whether a spin system contains any electrons. If so the function will return TRUE else it will return FALSE.

### Return Value:

An integer.

### Examples:

```
#include <gamma.h>
main()
  {
  AX2.spin_sys(3);                          // define spin system AX2 containing three spins.
  if(!AX2.electrons())
     cout << “\nNo electrons here”;         // This will print cause  all 1H by default.
```

```
    AX2.isotope(0,”e-”);                      // Set first spin to an electron
    if(CH3.electrons())
       cout << “\nElectrons are present”;     // AX2 is now has an electron spin, this will print.
    }
```

**See Also:**

## 3.4.21 nepair

## 3.4.22 enpair

**Usage:**

```
#include <HSLib/SpinSys.h>
int spin_sys::nepair(int i, int j) const
```

**Description:**

The function *nepair* is used to test whether the spins i and j are a nucleus and electron pair. If so the function will return TRUE else it will return FALSE.

**Return Value:**

An integer.

**Examples:**

```
  #include <gamma.h>
  main()
    {
    AX2.spin_sys(3);                          // define spin system AX2 containing three spins.
    AX2.isotope(0,”e-”);                      // Set first spin to an electron
    if(CH3.nepair(0,1))
       cout << “\nElectron-Nucleon pair”;     // AX2 is now has an electron spin, this will print.
    }
```

**See Also:**

## 3.4.23 pairidx

**Usage:**

```
#include <HSLib/SpinSys.h>
int spin_sys::pairidx(int i, int j) const
```

**Description:**

The function *pairidx* is used to produce a single index associated with the spin pair containing i and j. Ordering is 0 = (0,1) = (1,0); 1 = (0,2) = (2,0); 2 = (0,3) = (3,0); ...; NS = (1,2) = (2,1); NS+1 = (1,3) = (3,1); ....

Such indexing allows users to loop over spin pairs without redundancies..

**Return Value:**

An integer.

**Examples:**

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

```
main()
  {
  }
```

**See Also:**

### 3.4.24 isotopes

**Usage:**

```
#include <HSLib/SpinSys.h>
int spin_sys::isotopes() const
int spin_sys::isotopes(int i) const
int spin_sys::isotopes(const string& I) const
```

**Description:**

The function *isotpes* is used to count spins or spin isotopes in the system. Without arguments it will return the number of unique isotopes found in the spin system. With string I it will return the number of spin having the isotope type I. With an index i it will return the isotope symbol for this type, where i spans [0, # unique isotopes).

**Return Value:**

An integer.

**Examples:**

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

**See Also:**

# 3.5 Spin Flags Functions

## 3.5.1 SetFlag

**Usage:**

```
#include <HSLib/SpinSys.h>
void spin_sys::SetFlag(int spin, bool TF)
```

**Description:**

Function *SetFlag* is used to mark a particular spin as TRUE or FALSE. Used in combination with the function *SetFlags*, users can mark any group of spins in a spin system.

**Return Value:**

None.

**Example:**

```
#include <gamma.h>
main()
  {
  CH3.spin_sys(3);                        // Define spin system CH3 containing three spins.
  CH3.SetFlag(0,0 );                      // Set the first spin’s flags to false.
  CH3.SetFlag(2,1 );                      // Set the third spin’s flags to true.
  }
```

**See Also: SetFlags, GetFlag, GetFlags**

## 3.5.2 SetFlags

**Usage:**

```
#include <HSLib/SpinSys.h>
void spin_sys::SetFlags(bool TF)
void spin_sys::SetFlags(const string& iso, bool TF)
void spin_sys::SetFlags(const Isotope& I, bool TF)
```

**Description:**

Function *SetFlags* is used to mark sets of spins in a spin system to either be TRUE or FALSE. Which spins are marked depends upon any specified selectivity. With no selectivity all spin flags are set to the value ***TF***. All spins of a particular isotope may be set to ***TF*** if an isotope type is specified by either a string iso ("1H", "131Xe",...) or by an isotope I. This function is useful in creating functions/programs in which computations are performed only on specific spins (e.g. selectively pulsing a couple of spins in a spin system).

**Return Value:**

None.

**Example:**

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

```
  {
  CH3.spin_sys(3);                          // define spin system CH3 containing three spins.
  CH3.SetFlags(0);                          // Set all spin flags to false
  CH3.SetFlags("1H",1);                     // Set all proton spin flags to true
  }
```

**See Also: SetFlag, GetFlag, GetFlags**

## 3.5.3 GetFlag

**Usage:**

#include <HSLib/SpinSys.h>

void spin_sys::GetFlag (int) const

**Description:**

Function *GetFlag* is used to obtain a spin's flag setting. The spin will either be "TRUE" or "FALSE".

**Return Value:**

None.

**Examples:**

```
#include <gamma.h>
main()
  {
  CH3.spin_sys(3);                          // define spin system CH3 containing three spins.
  if(CH3.GetFlag(0))                        // see what the T/F status of spin 0 is
     cout << “\nSpin 0 is TRUE”;
  else
     cout << “\nSpin 0 is FALSE”;
  }
```

**See Also: flags**

## 3.5.4 GetFlags

**Usage:**

#include <HSLib/SpinSys.h>

vector<bool> spin_sys::GetFlags() const

vector<bool> spin_sys::GetFlags(bool TF) const

vector<bool> spin_sys::GetFlags(int spin, bool TF, bool DefTF) const

vector<bool> spin_sys::GetFlags(const string& isoin, bool TF, bool DefTF) const

**Description:**

Function *GetFlags* is used to obtain spin system flag settings. A vector of booleans the with elements for each spin will be returned. Without arguments the function returns a copy of the current system spin flags. With a True/False setting ***TF*** input the returned vector will have elements for each spin all set to ***TF***. When a selectivity is specified along with a default True/False state ***DefTF*** the returned vector will have elements for each spin all set to ***DefTF*** except for the slected spins which will be set to ***TF***.

**Return Value:**

None.

**Examples:**

```
#include <gamma.h>
main()
  {
  CH3.spin_sys(3);                          // define spin system CH3 containing three spins.
  if(CH3.flag(0))                           // see what the T/F status of spin 0 is
     cout << “\nSpin 0 is TRUE”;
  else
     cout << “\nSpin 0 is FALSE”;
  CH3.flag(0, TRUE);                        // spin 0 is now certainly marked as TRUE
  CH3.flag(“1H”, FALSE);                    // all protons in the system are marked as FALSE
  CH3.flag(CH3.isotope(2), TRUE);           // spins of same isotope type as spin 2 marked TRUE
  }
```

**See Also: flags**

## 3.5.5 name

**Usage:**

#include <HSLib/SpinSys.h>
void spin_sys::name(const string&)
const string& spin_sys::name()

**Description:**

*name* is used to specify or retrieve a spin system name.

**Return Value:**

None.

**Example:**

```
#include <gamma.h>
main()
  {
  spin_sys sys;                             // A new spin system
  sys.read("filein.sys");                   // Set from external file
  string sysname = sys.name();              // Retrieve system name
  cout << "\n\tSystem Name: " << sysname;// Print system name
  sys.name("newname");                      // Set name to new one
  }
```

**See Also:**

# 3.6 Input/Output

## 3.6.1 =

**Usage:**

```
#include <HSLib/SpinSys.h>
void operator= (p_set&, const spin_sys&)
void operator= (p_set&)
```

**Description:**

= is used here to equate a spin system to a parameter set or vice versa.

**Return Value:**

None.

**Example(s):**

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

**See Also:**

## 3.6.2 +=

**Usage:**

```
#include <HSLib/SpinSys.h>
void operator+= (p_set&, const spin_sys&)
```

**Description:**

*name* is used to specify or retrieve a spin system name.

**Return Value:**

None.

**Example(s):**

```
#include <spin_sys.h>
```

**See Also:**

## 3.6.3 write

**Usage:**

#include <HSLib/SpinSys.h>
void spin_sys::write(const string& filename);

**Description:**

The function *write* is enables one to specifically write the spin_sys out to a readable ASCII file. Spin system parameters are written in the standard parameter set format and readable by the function *read*.

**Return Value:**

None.

**Example:**

```
#include <gamma.h>
main()
  {
  spin_sys sys(3);                        // define spin_sys sys containing three spins.
  sys.write("test.sys");                  // write information in spin_sys sys to file test.sys.
  }
```

**See Also: read**

## 3.6.4 read

**Usage:**

#include <HSLib/SpinSys.h>
void spin_sys::read (const string& filename);

**Description:**

The function *read* enables one to read in the spin_sys from an external file. The file is assumed to be written in the standard parameter set format (see the Section Spin Syst Parameter Files later in this Chapter) utilizing the standard parameter names associated with a spin_sys. Typically, the file being read was generated from a previous simulation through the use of the analogous spin_system function *write*. Alternatively, one can use an editor to construct a parameter set file for the spin_sys.

**Return Value:**

None.

**Example(s):**

```
#include <gamma.h>
main()
  {
  spin_sys sys;                           // define an empty spin_sys.
  sys.read("test.sys");                   // read in the spin_sys from file test.sys.
  }
```

**See Also: write**

### 3.6.5 print

**Usage:**

#include <HSLib/SpinSys.h>
ostream& spin_sys::print(ostream&) const

**Description:**

The function *print* is used to print out all the current information stored in a spin system object. This includes the number of spins and spin isotope types.

**Return Value:**

None.

**Example(s):**

```
#include <gamma.h>
main()
  {
  CH3.spin_sys(3);                          // define spin system CH3 containing three spins.
  CH3.print( );                             // print out information in spin system CH3.
  }
```

### 3.6.6 <<

**Usage:**

#include <HSLib/SpinSys.h>
ostveream& operator<< (osteram&, spin_sys&)

**Description:**

The operator << is used for standard output of a spin system. This includes the number of spins and spin isotope types.

**Return Value:**

None.

**Example(s):**

```
#include <gamma.h>
main()
  {
  CH3.spin_sys(3);                          // define spin system CH3 containing three spins.
  cout << CH3;                              // write the spin system CH3 to standard output.
  }
```

**See Also: write, read, print**

## 3.7 Description

Two fundamental quantities of a spin_sys are the **number of spins** it contains and the intrinsic **spin angular momentum** (I) of each spin. These factors set the dimension of the associated spin **Hilbert space**.

Hilbert Space Definition

1. Number of Spins
2. Spin Quantum Numbers

Mathematically the Hilbert space for a single spin ***i*** is given by

$$HS(i) = 2I_i + 1 \tag{3-1}$$

where ***i*** is the spin label and $\boldsymbol{I_i}$ the associated spin quantum number. The total spin system Hilbert space (or composite Hilbert space) is the product of the individual spin Hilbert spaces. This is formally given by

$$HS = \prod_{i=1}^{nspins} [2I_i + 1] . \tag{3-2}$$

A core spin system[1] is created by the constructor function *spin_sys()*. At the time of construction, the number of spins may be specified, e.g. *spin_sys(#)*, where # is a positive integer representing the number of spins in the system. By default, each of these spins is assigned to be a proton (with a value I = 1/2). Individual spins may subsequently have their intrinsic spin angular momentum set directly with the function *isotope*.

The quantities inherent in eacy spin_sys are used to set up various Hilbert space operators, many of which are provided implicitly in GAMMA. As such, their values should be maintained as record of the parameters used in a simulation.

[1] Class spin_sys produces a “core spin system” because it is the base class from which several other spin systems are derived. A spin system from spin_sys contains only the bare essentials necessary to define a spin system.

## 3.8 Parameter Files

This section describes how an ASCII file may be constructed that is self readable by a spin_sys. The file can be created with an editor of the users choosing and is read with the spin_sys member function “read”. Examples of these files are given later in this chapter. Keep in mind that parameter ordering in the file is arbitrary. Some parameters are essential, some not. Also, other parameters are allowed in the file which do not relate to the spin system.

### Spin system name: SysName)

A spin system name may be entered. It has no mathematical function in GAMMA but comes in handy when quickly scanning the input file or tagging some output file with the spin system. This parameter is optional.

Parameter type 2 indicates a string parameter.

### Number of spins: Nspins

It is essential that the number of spins be specified. A simple integer is input here. This is the first thing that will be read from the file so keep it somewhere near the top.

### Isotope Types: Iso(i)

Spin isotopes are specified with this parameter. These are optional, spins for which there is no isotope specified will be set to protons. A complete lookup table of isotope information is internal to GAMMA and scanned upon spin system construction depending upon the isotope types. Isotopes are specified by the atomic number immediately followed (no blanks) by the atomic symbol.

**Table 1:**

| Parameter | Assumed Units | Examples<br>Parameter (Type) : Value - Statement |
|---|---|---|
| SysName | none | SysName (2) : Glycine - Spin System Name |
| NSpins | none | Nspins (0) : 5 - Number of Spins |
| Iso(i) | none | Iso(0) (2) : 19F - Spin Isotope Type<br>Iso(1) (2) : 3H - Spin Isotope Type<br>Iso(3) (2) : 13C - Spin Isotope Type |

Parameter type 2 indicates a string parameter.

Parameter type 0 indicates an integer parameter.

Note that the first spin is spin zero and the last spin is spin N-1 in a spin system with N spins. The values need not be in any order in the file nor even following each other. Parameter type 0 indicates an integer parameter value whereas parameter type 2 indicates a string parameter.

## 3.9 Example Parameter Files

Because spin_sys contains very little information it is many times easier in a GAMMA program to simply code in all spin_sys parameters directly. On the other hand, it is common that a user has many parameter files associated with more complex quantities (spin_system, tensors, etc.) which happens to contain spin_sys information. The more complicated file can be used equally well, the needed values simply scanned amongst the other parameters present in the file. Here we give examples of files which contain only spin_sys parameters.

Galactose

```
SysName (2) : galact - Name of the Spin System (galactose)
NSpins  (0) : 8 - Number of Spins in the System
Iso(0)  (2) : 1H- Spin Isotope Type
Iso(1)  (2) : 1H- Spin Isotope Type
Iso(2)  (2) : 1H- Spin Isotope Type
Iso(3)  (2) : 1H- Spin Isotope Type
Iso(4)  (2) : 1H- Spin Isotope Type
Iso(5)  (2) : 1H- Spin Isotope Type
Iso(6)  (2) : 1H- Spin Isotope Type
Iso(7)  (2) : 1H- Spin Isotope Type
```

## 3.10 Example Programs

This section contains example programs which were used to test spin_sys functions and/or generate some of the documentation. A brief explanation is included with each program and these are refered to in the Chapter text. They should be included in the GAMMA program package in the / gamma/tests/spin_sys subdirectory.

qStates.cc

```
/* qStates.cc ********************************************                   **
**                                                                           **
**  This program tests the spin_sys function                                 **
** qStates. Three spin systems are checked                                   **
** and their corresponding matrices sent                                     **
**  to both standard output and to FrameMaker                                **
** compatible MIF files.                                                     **
**                                                                           **
*****************************************************************/

  #include <gamma.h>

  main ()
    {
     spin_sys sys(2);                           // 2 spin system
     spin_sys sys1(3);                          // 3 spin system
     cout << “\n” << sys.qStates() << “\n”;     // Output for HS=4
     FM_Matrix(“mx1.mif”, sys.qStates());
     sys.isotope(1,”2H”);                       // Switch 2nd spin I=1
     cout << “\n” << sys.qStates() << “\n”;     // Output for HS=6
     FM_Matrix(“mx2.mif”, sys.qStates());
     cout << “\n” << sys1.qStates() << “\n”;    // Output for HS=8
     FM_Matrix(“mx3.mif”, sys1.qStates());
     cout << “\n”;
    }
```

## 3.11 List of Available Isotopes

***Available Isotopes in GAMMA***

| | Element | I | | Element | I | | Element | I |
|---|---|---|---|---|---|---|---|---|
| 1H | Hydrogen | 1/2 | 2H | Deuterium | 1 | 3H | Tritium | 1/2 |
| 3He | Helium | 1/2 | 6Li | Lithium | 1 | 7Li | Lithium | 3/2 |
| 9Be | Beryllium | 3/2 | 10B | Boron | 3 | 11B | Boron | 3/2 |
| 13C | Carbon | 1/2 | 14N | Nitrogen | 1 | 15N | Nitrogen | 1/2 |
| 17O | Oxygen | 5/2 | 19F | Fluorine | 1/2 | 21Ne | Neon | 3/2 |
| 23Na | Sodium | 3/2 | 25Mg | Magnesium | 5/2 | 27Al | Aluminum | 5/2 |
| 29Si | Silicon | 1/2 | 31P | Phosphorus | 1/2 | 33S | Sulfur | 3/2 |
| 35Cl | Chlorine | 3/2 | 37Cl | Chlorine | 3/2 | 39K | Potassium | 3/2 |
| 41K | Potassium | 3/2 | 43Ca | Calcium | 7/2 | 45Sc | Scandium | 7/2 |
| 47Ti | Titanium | 5/2 | 49Ti | Titanium | 7/2 | 50V | Vanadium | 6 |
| 51V | Vanadium | 7/2 | 53Cr | Chromium | 3/2 | 55Mn | Manganese | 5/2 |
| 57Fe | Iron | 1/2 | 59Co | Cobalt | 7/2 | 61Ni | Nickel | 3/2 |
| 63Cu | Copper | 3/2 | 65Cu | Copper | 3/2 | 67Zn | Zinc | 5/2 |
| 69Ga | Gallium | 3/2 | 71Ga | Gallium | 3/2 | 73Ge | Germanium | 9/2 |
| 75As | Arsenic | 3/2 | 77Se | Selenium | 1/2 | 79Br | Bromine | 3/2 |
| 81Br | Bromine | 3/2 | 85Rb | Rubidium | 5/2 | 87Rb | Rubidium | 3/2 |
| 87Sr | Strontium | 9/2 | 89Y | Yttrium | 1/2 | 91Zr | Zirconium | 5/2 |
| 93Nb | Niobium | 9/2 | 95Mo | Molybdenum | 5/2 | 97Mo | Molybdenum | 5/2 |
| 99Tc | Technetium | 9/2 | 99Ru | Ruthenium | 5/2 | 101Ru | Ruthenium | 5/2 |
| 103Rh | Rhodium | 1/2 | 105Pd | Palladium | 5/2 | 107 | Silver | 1/2 |
| 109Ag | Silver | 1/2 | 111Cd | Cadmium | 1/2 | 113Cd | Cadmium | 1/2 |
| 113In | Indium | 9/2 | 115In | Indium | 9/2 | 117Sn | Tin | 1/2 |
| 119Sn | Tin | 1/2 | 121Sb | Antimony | 5/2 | 123Sb | Antimony | 7/2 |
| 123Te | Tellurium | 1/2 | 125Te | Tellurium | 1/2 | 127I | Iodine | 5/2 |
| 129Xe | Xenon | 1/2 | 131Xe | Xenon | 3/2 | 133Cs | Cesium | 7/2 |
| 135Ba | Barium | 3/2 | 137Ba | Barium | 3/2 | 138La | Lanthanum | 5 |
| 139La | Lanthanum | 7/2 | 141Pr | Praseodymium | 5/2 | 143Nd | Neodymium | 7/2 |
| 145Nd | Neodymium | 7/2 | 147 | Samarium | 7/2 | 149Sm | Samarium | 7/2 |
| 151Eu | Europium | 5/2 | 153Eu | Europium | 5/2 | 155Gd | Gadolinium | 3/2 |
| 157Gd | Gadolinium | 3/2 | 159Tb | Terbium | 3/2 | 161Dy | Dysprosium | 5/2 |
| 163Dy | Dysprosium | 5/2 | 165Ho | Holmium | 7/2 | 167Er | Erbium | 7/2 |
| 169Tm | Thulium | 1/2 | 171Yb | Ytterbium | 1/2 | 173Yb | Ytterbium | 5/2 |
| 175Lu | Lutetium | 7/2 | 176Lu | Lutetium | 7 | 177Hf | Hafnium | 7/2 |
| 179Hf | Hafnium | 9/2 | 181Ta | Tantalum | 7/2 | 183W | Tungsten | 1/2 |
| 185Re | Rhenium | 5/2 | 187Re | Rhenium | 5/2 | 187Os | Osmium | 1/2 |
| 189Os | Osmium | 3/2 | 191Ir | Iridium | 3/2 | 193Ir | Iridium | 3/2 |
| 195Pt | Platinum | 1/2 | 197Au | Gold | 3/2 | 199Hg | Mercury | 1/2 |
| 201Hg | Mercury | 3/2 | 203Tl | Thallium | 1/2 | 205Tl | Thallium | 1/2 |
| 207Pb | Lead | 1/2 | 209Bi | Bismuth | 9/2 | 235U | Uranium | 7/2 |
| e- | Electron | 1/2 | | | | | | |

# 4 Spin Operators

## 4.1 Overview

The module *spin_op (spin operator*) contains routines that produce operators based on spin angular momentum.

All spin operators are inherently associated with either a single spin or group of spins. Thus there are fundamental ties between each spin_op (spin operator) and some spin_sys (spin system)[1]. Although it is not mandatory that a spin system be specified when declaring a new spin operator, most spin operator functions must have a spin system given as an argument during the function call.

Each spin operators can (and may) be stored in two ways. When possible (for all product operators) the product operator factors belonging to each spin in the spin operator are stored separately. All other spin operators are stored in the form of a full Hilbert space matrix. All storage is dynamic using the C++ free memory.

The produced spin operator matrices are invariably returned in the product basis. See the detailed description of class *operator* and *basis* for details on various operator bases.

## 4.2 Available Spin Operator Functions

### Spin Operator Basic Functions

| | | |
|---|---|---|
| spin_op | - Constructor | page 60 |
| ~ | - Destructor | page 60 |
| = | - Assignment | page 61 |
| + | - Addition | page 62 |
| += | - Unary sddition | page 60 |
| - | - Subtraction | page 63 |
| -= | - Unary subtraction | page 63 |
| * | - Multiplication | page 64 |
| *= | - Unary multiplication | page 63 |
| / | - Division | page 65 |
| /= | - Unary division | page 63 |
| << | - Spin operator output | page 65 |

### Single Spin Operator Functions

| | | |
|---|---|---|
| Ie | - identity operator (completes the basis set) | page 66 |
| Ix | - x component of spin angular momentum) | page 67 |
| Iy | - y component of spin angular momentum) | page 68 |

[1] Note that *spin_sys* is a base class for several spin systems. All functions in this chapter which use *spin_sys* as an argument may equally well use one of the derived classes, *e.g. spin_system*, *molecule*, ..., etc.

| | | |
|---|---|---|
| Iz | - z component of spin angular momentum) | page 70 |
| Ip | - + component of spin angular momentum) | page 72 |
| Im | - - component of spin angular momentum) | page 73 |
| Ia | - polarization operator) | page 75 |
| Ib | - polarization operator) | page 76 |
| Ipol | - polarization operator) | page 77 |

## Multiple Spin Operator Functions

| | | |
|---|---|---|
| Fe | - identity operator (completes the basis set) | page 78 |
| Fx | - x component of spin angular momentum) | page 79 |
| Fy | - y component of spin angular momentum) | page 80 |
| Fz | - z component of spin angular momentum) | page 81 |
| Fp | - + component of spin angular momentum) | page 82 |
| Fm | - - component of spin angular momentum) | page 83 |
| Fa | - polarization operator) | page 84 |
| Fb | - polarization operator) | page 85 |
| Fpol | - polarization operator) | page 86 |
| Fpdt | - product operator) | page 86 |

## Rotation Operator Functions

| | | |
|---|---|---|
| Rx | - Rotations about the x-axis | page 88 |
| Ry | - Rotations about the y-axis | page 91 |
| Rz | - Rotations about the z-axis | page 92 |
| Rxy | - Rotations in the xy-plane | page 94 |
| Ryz | - Rotations in the yz-plane | page 96 |
| Rzx | - Rotations in the zx-plane | page 97 |
| Rxyz | - Rotations about any axis | page 99 |
| R_Euler | - Euler rotations about any axis | page 100 |
| Rspace | - Rotations about an axis in a specified plane | |
| Raxis | - Rotations about a specified axis in the xy-plane | page 94 |
| Rplane | - Rotations about an axis in a specified plane | |

# 4.3 Routines

## 4.3.1 spin_op

**Usage:**

```
#include <HSLib/SpinOp.h>
spin_op ()
spin_op (spin_sys &sys)
spin_op (spin_op& SOp)
```

**Description:**

The function ***spin_op*** is used to create a spin operator quantity. There are currently three methods for creating a new spin operator.

1. spin_op () - When ***spin_op*** is invoked without arguments it creates an empty spin operator. In further manipulations, typically an assignment to some other spin operator, it will be associated with some spin system.
2. spin_op (spin_sys &sys) - When ***spin_op*** is invoked with a spin system ***sys*** as the argument it creates an empty spin operator associated with the spin system given. The resultant spin operator can be used with all spin operator functions.
3. spin_op(spin_op &SOp) - When ***spin_op*** is used with another spin operator ***SOp*** as its argument a new spin operator is produced that is identical to the spin operator given.

**Return Value:**

The function (constructor) ***spin_op*** returns no parameters. It is used strictly to create a new spin operator.

**Examples:**

```
#include <gamma.h>
main()
  {
  spin_op SOp;                          // new empty spin operator.
  spin_sys ABX(3);                      // create a three spin system called ABX.
  spin_op SOp1(ABX);                    // new spin operator associated with system ABX.
  spin_op SOp2(SOp);                    // new SOp2, copied from SOp1 (sys = ABX).
  SOp = SOp1;                           // now SOp is equal to SOP1 also.
  }
```

**See Also:** =

## 4.3.2 =

### Usage:

```
#include <HSLib/SpinOp.h>
spin_op operator = (spin_op &SOp)
spin_op operator = (matrix& mx)
matrix operator = (spin_op &SOp)
gen_op operator = (spin_op &SOp)
```

### Description:

The assignment ***operator =*** provides the use of equality in the algebraic manipulations of spin operators. The user may set one spin operator equal to another spin operator. It is also allowed (but not advantageous) to assign a matrix to a spin operator and to assign a spin operator to a matrix or general operator.

### 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_op SOp;                       // create a new spin operator SOp
  SOp = Fx(AX);                      // set SOp to spin operator Fx for the AX system.
  matrix mx;                         // create a new matrix.
  mx = SOp;                          // Set matrix mx to the current SOp.
  gen_op Op;                         // create a new general operator.
  Op = SOp;                          // Set operator Op to the current SOp.
  }
```

**See Also: N**one

## 4.3.3 +

### Usage:

```
#include <HSLib/SpinOp.h>
spin_op operator + (spin_op& SOp1, spin_op& SOp2)
gen_op operator + (gen_op& Op, spin_op& SOp)
gen_op operator + (spin_op& SOp, gen_op& Op)
```

### Description:

This allows for the addition of two spin operators, **SOp1** + **SOp2** and the addition of a spin operator to a general operator **Op**.

1. SOp1 + SOp2 - Definition of the addition of two spin operators **SOp1** and **SOp2**. A check is made to insure that both operators are connected to the same spin system.
2. Op +SOp - Definition of the addition of an operator **Op** to a spin operator **SOp**. Since spin operators are always stored in the product basis (default basis) the addition will produce a new operator in the product basis. If **Op** is not in the default basis, it will be transformed into it prior to the addition. The result is *new*

*operator in the product (default) basis.*

3. SOp +Op - Same effect as 2. above.

**Return Value:**

A new spin operator or a new operator depending upon the addition requested.

**Example(s):**

```
#include <gamma.h>
main()
  {
  spin_sys AX(2);                     // create a two spin system called AX.
  spin_op SOp;                        // create a new spin operator SOp.
  SOp = Fx(AX) + Fy(AX);              // set SOp equal to the Fx + Fy for the AX system.
  gen_op Op;                          // create a new general operator.
  Op = Hcs;                           // Set Op to the isotropic chemical shift Hamiltonian.
  Op = Op + Fx(AX);                   // Add Fx to the general operator Op.
  }
```

**See Also:**

-, +=

## 4.3.4 +=

**Usage:**

```
#include <HSLib/SpinOp.h>
void spin_op += (spin_op &SOp)
```

**Description:**

This allows for the addition of two spin operators of the type **SOp1 = SOp1** + **SOp2**. A check is made to insure that both spin operators are associated with the same spin system. Use of this operation is more computationally efficient that the two step operation.That is, the statement **SOp1** += **SOp2**; is preferred over the statement **SOp1** = **SOp1** + **SOp2**;. The function *produces a result in the product basis*.

**Return Value:**

A new operator which exists in an appropriate representation.

**Example(s):**

```
#include <gamma.h>
main()
  {
  spin_sys thiophene(4);
  spin_op SOp1(thiophene), SOp2(thiophene); // define two operators SOp1 and SOp2.
  SOp1 += SOp2;                       // set operator SOp1 to the sum of SOp1 + SOp2.
  }
```

**See Also:**

+,-

## 4.3.5 -

**Usage:**

```
#include <HSLib/SpinOp.h>
spin_op operator - (spin_op& SOp1, spin_op& SOp2)
gen_op operator - (gen_op& Op, spin_op & SOp)
gen_op operator - (spin_op& SOp, gen_op& Op)
```

**Description:**

This allows for the subtraction of two spin operators, **SOp1** + **SOp2** and the subtractions involving spin operators with operators.

1. SOp1- SOp2 - Definition of the subtraction of two spin operators **SOp1** and **SOp2**. A check is made to insure that both operators are connected to the same spin system.
2. Op - SOp - Definition of the subtraction of a spin operator **SOp** from an operator **Op**. Since spin operators are always stored in the product basis (default basis) the subtraction will produce a new operator in the product basis. If **Op** is not in the default basis, it will be transformed into it prior to the operation. The result is *new operator in the product (default) basis*.
3. SOp - Op - Definition of the subtraction of an operator **Op** from a spin operator **SOp**. Since spin operators are always stored in the product basis (default basis) the subtraction will produce a new operator in the product basis. If **Op** is not in the default basis, it will be transformed into it prior to the operation. The result is *new operator in the product (default) basis*.

**Example(s):**

```
#include <gamma.h>
main()
  {
  spin_sys AX(2);                    // create a two spin system called AX.
  spin_op SOp;                       // create a new spin operator SOp.
  SOp = Fx(AX) - Fy(AX);             // set SOp equal to the Fx - Fy for the AX system.
  gen_op Op, Op2;                    // create two new general operators.
  Op = Op1 - Fx(AX);                 // Subtract Fx from the general operator Op1.
  }
```

**See Also:**+, *, /

## 4.3.6 -=

**Usage:**

```
#include <HSLib/SpinOp.h>
spin_op operator -= (spin_op& SOp)
gen_op operator -= (spin_op& SOp) ***???? this is not possible SOSI
```

**Description:**

This allows for the subtraction of two spin operators of the type **SOp1 = SOp1 - SOp2**. A check is made to insure that both spin operators share the same spin system Use of this operator is more computationally efficient that the two step operation.That is, the statement **SOp1** -= **SOp2**; is preferred over the statement **SOp1**

= **SOp1** - **SOp2**;.

**Return Value:**

A modified spin operator which exists in the product basis.

**Example(s):**

```
#include <gamma.h>
main()
  {
  matrix m;
  spin_op SOp1,SOp2;                    // define two spin operators SOp1 and SOp2.
  SOp1 -= SOp2;                         // subtract SOp2 from SOp1.
  }
```

**See Also: +, +=, -**

### 4.3.7 *

**Usage:**

```
#include <HSLib/SpinOp.h>
spin_op operator * (spin_op& SOp1, spin_op& SOp2)
gen_op operator * (spin_op& SOp, gen_op& Op)
gen_op operator * (gen_op &Op, spin_op& SOp)
spin_op operator * (complex& c, spin_op& SOp)
spin_op operator * (spin_op& SOp, complex& c)
```

**Description:**

This allows for the multiplication of two spin operators **SOp1** and **SOp2**, the multiplication of an operator and a matrix, and for the multiplication of a scalar and an operator.

1. For the multiplication of two spin operators, a check is made to insure that both are applicable to the same spin system. Here, the order of the operators can make a difference in the result.
2. For the multiplication of a spin operator times an operator, an operator is produced. Since the spin operator is in the product (default) basis, it is insured so that operator **SOp** is changed into the default basis before the multiplication.
3. The multiplication of a matrix time an operator also produces an operator. The treatment is similar to the previous usage except the ordering can make a difference.
4. For the multiplication of a scalar times an operator, the scalar c is multiplied into each element of **SOp** to produce an operator in the same basis of **SOp.**
5. The multiplication of an operator times a scalar produces the same result as multiplication of a scalar times an operator.

**Return Value:**

**Example(s):**

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

```
complex x;
spin_op SOp1,SOp2,SOp3; // define three operators SOp1,SOp2, and SOp3.
}
```

**See Also:**

+, -, /

## 4.3.8 /

**Usage:**

```
#include <HSLib/SpinOp.h>
gen_op operator / (gen_op& a, gen_op& b)
gen_op operator / (complex& c, gen_op& a)
gen_op operator / (gen_op& a, complex& c)
```

**Description:**

1. This is not currently implemented

**Return Value:**

## 4.3.9 <<

**Usage:**

```
#include <HSLib/SpinOp.h>
ostream& operator << (ostream& a, const spin_op &SOp)
```

**Description:**

The function (operation) << is provided for the standard output of a spin operator.

**Return Value:**

The function returns an output stream which prints the spin operator matrix.

**Example(s):**

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

**See Also:**

# 4.4 Single-Spin Spin Operators

## 4.4.1 Ie

**Usage:**

```
#include <HSLib/SpinOp.h>
spin_op Ie(spin_sys &sys, int spin);
```

**Description:**

The function ***Ie*** provides a “spin operator” which is merely the identity matrix. This is not particularly useful as a stand alone function but is essential for the maintenance of the *Class spin_op*. The function returns a spin operator as a matrix in the product (default) basis spanning the full composite Hilbert space of the spin system given as an argument. The specified spin is of no consequence external to the class.

**Return Value:**

The matrix representation of the operator in the composite Hilbert space of the spin system.

**Examples:**

```
#include <gamma.h>
main()
  {
  spin_sys ab(2);                           // Construct a 2 spin system called ab
  spin_op SOp = Ie(ab,0);                   // SOp set to Ie for the first spin of spin system ab.
  SOp = Ie(ab,1);                           // SOp reset to Ie for the second spin of system ab.
  }
```

**See Also: Ix, Iy, Iz**

**Mathematical Basis:**

There are four operators which provide a basis for spin angular momentum, the set $\{\mathbf{I_e}, \mathbf{I_x}, \mathbf{I_y}, \mathbf{I_z}\}$. Although ***Ie*** is trivial (being simply the identity matrix) its provision is essential in order to complete the basis mentioned above. The matrix representation of $\boldsymbol{I_e}$ for a single spin ***i***, $\boldsymbol{I_{ie}}$, is efficiently expressed in the dimension of the Hilbert space of the spin itself, spanning $2(I_i+1)$. Examples are shown below for particles with spin 1/2, 1, and 3/2 respectively.

$$\boldsymbol{I}_e\left(I = \frac{1}{2}\right) = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \qquad \boldsymbol{I}_e(I = 1) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad \boldsymbol{I}_e\left(I = \frac{3}{2}\right) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Note that the matrix forms above would be output (printed) by GAMMA only for a spin system with one spin. For a multispin system the printed form is that which spans the composite Hilbert space of the entire spin system, as obtained from direct products of these forms as specified in equation requirenment (18-103). The $\boldsymbol{I_{ie}}$ operators are used most frequently in the expansion of other spin operators into their composite Hilbert space forms through use of these direct products.

## 4.4.2 Ix

### Usage:

```
#include <HSLib/SpinOp.h>
spin_op Ix(spin_sys &sys, int spin);
```

### Description:

The function ***Ix*** produces the operator for the x-component of spin angular momentum, $\boldsymbol{I}_{ix}$, for the specified spin of the spin system as given in the arguments. The result is a spin operator as a matrix in the product (default) basis spanning the composite space of the spin system. The argument parameters are the spin system, *sys*, and the specific spin in that system, ***spin***.

### Return Value:

The matrix representation of the operator $\boldsymbol{I}_{ix}$ in the composite Hilbert space of the spin system.

### Example(s):

```
#include <gamma.h>
main()
  {
  spin_sys ab(2);                          // Declare a two spin system ab
  spin_op SOp = Ix(ab,0);                  // Declare and set SOp to Ix for the 1st spin in ab.
  SOp=Ix(ab,1);                            // Reset SOp to Ix for the second spin in ab.
  }
```

### See Also: Ie, Iy, Iz, Fx

### Mathematical Basis:

$$\boldsymbol{I}_x|\alpha\rangle = \frac{1}{2}(\boldsymbol{I}_+ + \boldsymbol{I}_-)|\alpha\rangle = \frac{1}{2}|\beta\rangle \qquad \boldsymbol{I}_x|\beta\rangle = \frac{1}{2}(\boldsymbol{I}_+ + \boldsymbol{I}_-)|\beta\rangle = \frac{1}{2}|\alpha\rangle \tag{18-4}$$

The eigenkets of $\boldsymbol{I}_x$ for a spin 1/2 particle are[1]

$$|+_x\rangle = \frac{1}{\sqrt{2}}[|\alpha\rangle + |\beta\rangle] \quad \text{and} \quad |-_x\rangle = \frac{1}{\sqrt{2}}[-|\alpha\rangle + |\beta\rangle] \quad . \tag{18-5}$$

These are easily obtained from rotations of the eigenkets of $\boldsymbol{I}_z$ about the y-axis by $\pi/2$, that is,

$$\boldsymbol{R}_y^{(1/2)}\left(\frac{\pi}{2}\right)|\alpha\rangle = |+_x\rangle \qquad \boldsymbol{R}_y^{(1/2)}\left(\frac{\pi}{2}\right)|\beta\rangle = |-_x\rangle \tag{18-6}$$

In general, the matrix elements of the operator are given by[2]

$$\langle m_i|\boldsymbol{I}_{ix}|n_i\rangle = \frac{1}{2}[I_i(I_i+1) - m_i(m_i \pm 1)]^{1/2}\delta_{m_i,\, n_i \pm 1} \tag{18-7}$$

The matrix representation of $\boldsymbol{I}_x$ for a single spin i, $\boldsymbol{I}_{ix}$, is efficiently expressed in the dimension of the Hilbert

[1] Goldman, pgs. 66-67.
[2] Goldman, page 73, equation (3.93).

space of the spin itself, spanning 2($I_i$+1). Examples are shown below for a spin 1/2, 1, and3/2 particles[1].

$$\boldsymbol{I}_{ix}(I_i = 1/2) = \frac{1}{2}\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \qquad \boldsymbol{I}_{ix}(I_i = 1) = \frac{1}{\sqrt{2}}\begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \qquad \boldsymbol{I}_{ix}\left(I_i = \frac{3}{2}\right) = \frac{1}{2}\begin{bmatrix} 0 & \sqrt{3} & 0 & 0 \\ \sqrt{3} & 0 & 2 & 0 \\ 0 & 2 & 0 & \sqrt{3} \\ 0 & 0 & \sqrt{3} & 0 \end{bmatrix} \qquad (18\text{-}8)$$

Note that the matrices shown in equation (18-8) are in the single spin space only. In a multiple spin system, $\boldsymbol{I}_{ix}$ arrays in the composite Hilbert space can be determined by taking the appropriate direct products. As an example, if we consider a two spin system composed of a spin 1 and a spin 1/2 particle we have the following matrices.

$$\boldsymbol{I}_{1x} = \boldsymbol{I}_{1x} \otimes \boldsymbol{I}_{2e} = \frac{1}{\sqrt{2}}\begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \otimes \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = \frac{1}{\sqrt{2}}\begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

$$\boldsymbol{I}_{2x} = \boldsymbol{I}_{1e} \otimes \boldsymbol{I}_{2x} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \otimes \frac{1}{2}\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} = \frac{1}{2}\begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix} \qquad (18\text{-}9)$$

Other important relationships involving this operator are the following.

$$\boldsymbol{I}_x = \frac{1}{2}(\boldsymbol{I}_+ + \boldsymbol{I}_-) \qquad [\boldsymbol{I}_x, \boldsymbol{I}_y] = i\boldsymbol{I}_z \qquad [\boldsymbol{I}_x, \boldsymbol{I}_z] = -i\boldsymbol{I}_y$$

## 4.4.3 Iy

### Usage:

```
#include <HSLib/SpinOp.h>
spin_op Iy(spin_sys &sys, int spin);
```

### Description:

The function *I*y produces the y-component of spin angular momentum $\boldsymbol{I}_y$ for the specified spin of the spin system given in the argument. The result is a spin operator as a matrix in the product (default) basis spanning the composite space of the spin system. The argument parameters are the spin system, “sys”, and the specific

[1] These leave out the units of hbar, see Schiff, page 203.

spin in that system, “spin”.

### Return Value:

The matrix representation of the operator $\boldsymbol{I}_{iy}$ in the composite Hilbert space of the spin system.

### Examples:

```
#include <gamma.h>
main()
  {
  spin_sys ab(2);
  spin_op SOp(ab);
  SOp=Iy(ab,0);                         // SOp set to Iy for the first spin in the spin system.
  SOp=Iy(ab,1);                         // reset SOp to y component of second spin.
  }
```

### See Also: Ie, Ix, Iz, Fx

### Mathematical Basis:

The eigenkets of $\boldsymbol{I}_y$ for a spin 1/2 particle are[1]

$$|+_y\rangle = \frac{1}{\sqrt{2}}[|\alpha\rangle + i|\beta\rangle] \qquad \text{and} \qquad |\text{-}_y\rangle = \frac{1}{\sqrt{2}}[i|\alpha\rangle - |\beta\rangle] \tag{18-10}$$

The elements of the matrix *Iy* are obtained from the formula[2]

$$\langle m_i|\boldsymbol{I}_{iy}|n_i\rangle = \left(\pm\frac{i}{2}\right)[I_i(I_i+1) - m_i(m_i \pm 1)]^{1/2}\delta_{m_i,\, n_i \pm 1} \tag{18-11}$$

The function *Iy *****??????*

$$\boldsymbol{I}_y|\alpha\rangle = -\frac{i}{2}(\boldsymbol{I}_+ - \boldsymbol{I}_-)|\alpha\rangle = \frac{i}{2}|\beta\rangle \tag{18-12}$$

$$\boldsymbol{I}_y|\beta\rangle = -\frac{i}{2}(\boldsymbol{I}_+ - \boldsymbol{I}_-)|\beta\rangle = -\frac{i}{2}|\alpha\rangle \tag{18-13}$$

The matrix representation of $\mathbf{I}_y$ for a single spin i**,** $\mathbf{I}_{iy}$**,** is efficiently expressed in the dimension of the Hilbert space of the spin itself, spanning $2(I_i+1)$. Examples are shown below for a spin 1/2, 1, and3/2 particles[3].

$$\boldsymbol{I}_{iy}\left(I_i = \frac{1}{2}\right) = \frac{i}{2}\begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \quad \boldsymbol{I}_{iy}(I_i = 1) = \frac{i}{\sqrt{2}}\begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix} \quad \boldsymbol{I}_{iy}\left(I_i = \frac{3}{2}\right) = \frac{1}{2}\begin{bmatrix} 0 & -i\sqrt{3} & 0 & 0 \\ i\sqrt{3} & 0 & -2i & 0 \\ 0 & 2i & 0 & -\sqrt{3} \\ 0 & 0 & i\sqrt{3} & 0 \end{bmatrix} \tag{18-14}$$

Keep in mind that the matrices shown in equation (18-14) are in the single spin space only. In a multiple spin

[1] Goldman, pg. 67, equation (3.68).

[2] Goldman, page 73, equation (3.94).

[3] These leave out the units of hbar, see Schiff, page 203.

system, $\boldsymbol{I}_{iy}$ arrays in the composite Hilbert space can be determined by taking the appropriate direct products. For example, consider a two spin system composed of spin 1/2 particles. We would then have the following matrices.

$$\boldsymbol{I}_{1y} = \boldsymbol{I}_{1y} \otimes \boldsymbol{I}_{2e} = \frac{i}{2}\begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \otimes \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = \frac{i}{2}\begin{bmatrix} 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & -1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \tag{18-15}$$

$$\boldsymbol{I}_{2y} = \boldsymbol{I}_{1e} \otimes \boldsymbol{I}_{2y} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \otimes \frac{i}{2}\begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} = \frac{i}{2}\begin{bmatrix} 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{18-16}$$

Other important relationships involving this operator are the following.

$$\boldsymbol{I}_y = \frac{i}{2}(\boldsymbol{I}_+ - \boldsymbol{I}_-) \tag{18-17}$$

$$[\boldsymbol{I}_y, \boldsymbol{I}_x] = -i\boldsymbol{I}_z \qquad [\boldsymbol{I}_y, \boldsymbol{I}_z] = i\boldsymbol{I}_x \tag{18-18}$$

## 4.4.4 Iz

### Usage:

```
#include <HSLib/SpinOp.h>
spin_op Iz(spin_sys &sys, int spin);
```

### Description:

The function *Iz* produces the spin operator $\mathbf{I}_z$ for a specified single spin in a spin system, namely $\mathbf{I}_{iz}$. A spin operator as a matrix in the product (default) basis spanning the composite Hilbert space of the spin system is returned. The argument parameters taken are the spin system “sys” and the spin in that system “spin”.

### Return Value:

The matrix representation of the operator.

### Example(s):

```
#include <gamma.h>
main()
  {
  spin_sys ab(2);
  spin_op SOp(ab);
  SOp=Fz(ab);                          // SOp set to Iz of first spin of ab
  SOp=Fz(ab);                          // SOp reset to Iz for the second spin
  }
```

### See Also: Ie, Ix, Iy, Fz

### Mathematical Basis:

The spin operator $\boldsymbol{I}_{iz}$ has the property that

$$\boldsymbol{I}_{iz}|m_i\rangle = m|m_i\rangle \tag{18-19}$$

where $|m_i\rangle$ is a basis function for spin i with spin quantum number $m$. For a spin 1/2 particle the two possible spin states are $|\alpha\rangle$ and $|\beta\rangle$, and the following relationships hold.

$$\boldsymbol{I}_z|\alpha\rangle = \frac{1}{2}|\alpha\rangle \qquad \boldsymbol{I}_z|\beta\rangle = -\frac{1}{2}|\beta\rangle \tag{18-20}$$

From equation (18-19), the individual matrix elements of the operator are

$$\langle n_i|\boldsymbol{I}_{iz}|m_i\rangle = m\delta_{m_i n_i} = [I(1-\alpha)]\delta_{m_i n_i} \quad \text{where } \alpha \text{ spans } [1, 2(I+1)] \tag{18-21}$$

The matrix representation of $\boldsymbol{I}_z$ for a single spin i, $\boldsymbol{I}_{iz}$, is efficiently expressed in the dimension of the Hilbert space of the spin itself, spanning 2($I_i$+1). Examples are shown below for a spin 1/2, 1, and3/2 particles[1].

$$\boldsymbol{I}_{iz}(I_i = 1/2) = \frac{1}{2}\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \qquad \boldsymbol{I}_{iz}(I_i = 1) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & -1 \end{bmatrix} \qquad \boldsymbol{I}_{iz}\left(I_i = \frac{3}{2}\right) = \frac{1}{2}\begin{vmatrix} 3 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & -3 \end{vmatrix} \tag{18-22}$$

Note that the matrices shown in equation (18-22) are in the single spin space only. In a multiple spin system, $\boldsymbol{I}_{iz}$ arrays in the composite Hilbert space can be determined by taking the appropriate direct products. For example, consider a two spin system composed of a spin 1 and a spin 1/2 particle. We have the following matrices.

$$\boldsymbol{I}_{1z} = \boldsymbol{I}_{1z} \otimes \boldsymbol{I}_{2e} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & -1 \end{bmatrix} \otimes \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 \end{bmatrix} \tag{18-23}$$

$$\boldsymbol{I}_{2z} = \boldsymbol{I}_{1e} \otimes \boldsymbol{I}_{2z} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \otimes \frac{1}{2}\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} = \frac{1}{2}\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 \end{bmatrix} \tag{18-24}$$

Other useful relationships involving $\boldsymbol{I}_{iz}$ are the following[2].

$$[\boldsymbol{I}_{iz}, \boldsymbol{I}_{ix}] = i\boldsymbol{I}_{iy}, \qquad [\boldsymbol{I}_{iz}, \boldsymbol{I}_{i+}] = \boldsymbol{I}_{i+} \qquad [\boldsymbol{I}_{iz}, \boldsymbol{I}_{i-}] = \boldsymbol{I}_{i-} \tag{18-25}$$

The equations below demonstrate that equation (18-19) is satisfied for the matrix forms in (18-22) for the single spin case with I = 1/2 and with I = 1.

[1] These leave out the units of hbar, see Schiff, page 203.

[2] See Goldman, p. 61-62.

$$\boldsymbol{I}_{iz}|\alpha\rangle = \frac{1}{2}\begin{bmatrix}1 & 0\\0 & -1\end{bmatrix}\begin{bmatrix}1\\0\end{bmatrix} = \frac{1}{2}\begin{bmatrix}1\\0\end{bmatrix} = \frac{1}{2}|\alpha\rangle \qquad \boldsymbol{I}_{iz}|\beta\rangle = \frac{1}{2}\begin{bmatrix}1 & 0\\0 & -1\end{bmatrix}\begin{bmatrix}0\\1\end{bmatrix} = \frac{1}{2}\begin{bmatrix}0\\-1\end{bmatrix} = \left(-\frac{1}{2}\right)|\beta\rangle \quad (18\text{-}26)$$

$$\boldsymbol{I}_z|\alpha\rangle = \begin{bmatrix}1 & 0 & 0\\0 & 1 & 0\\0 & 0 & -1\end{bmatrix}\begin{bmatrix}1\\0\\0\end{bmatrix} = \begin{bmatrix}1\\0\\0\end{bmatrix} = 1|\alpha\rangle \qquad \boldsymbol{I}_z|\beta\rangle = \begin{bmatrix}1 & 0 & 0\\0 & 0 & 0\\0 & 0 & -1\end{bmatrix}\begin{bmatrix}0\\1\\0\end{bmatrix} = \begin{bmatrix}0\\0\\0\end{bmatrix} = 0|\beta\rangle$$

$$\boldsymbol{I}_z|\gamma\rangle = \begin{bmatrix}1 & 0 & 0\\0 & 1 & 0\\0 & 0 & -1\end{bmatrix}\begin{bmatrix}0\\0\\1\end{bmatrix} = \begin{bmatrix}0\\0\\-1\end{bmatrix} = -1|\gamma\rangle \quad (18\text{-}27)$$

## 4.4.5 Ip

**Usage:**

```
#include <HSLib/SpinOp.h>
spin_op Ip (spin_sys &sys, int spin);
```

**Description:**

The function *Ip* provides direct access to the single spin raising operators $\boldsymbol{I}_{i+}$. The function result is a spin operator as a matrix in the product (default) basis for the spin.

**Return Value:**

The matrix representation of the raising operator for a specific spin in a spin system.

**Example(s):**

```
#include <gamma.h>
main()
  {
  spin_sys ab(2);
  spin_op SOp(ab);
  SOp =Ip(ab, 0);                                // set SOp to I+ of the first spin in spin system ab.
  SOp =Ip(ab, 1);                                // reset SOp to raising operator for the second spin.
  }
```

**See Also: Im, Fp**

**Mathematical Basis:**

The raising operator $\boldsymbol{I}_+$ is defined to work on state $|I, m\rangle$ as[1]

$$\boldsymbol{I}_+|I, m\rangle = [I(I+1) - m(m+1)]^{1/2}|I, m+1\rangle \quad , \quad (18\text{-}28)$$

having matrix elements

$$\langle I, m'|\boldsymbol{I}_+|I, m\rangle = [I(I+1) - m(m+1)]^{1/2}\delta_{m+1, m'} \quad . \quad (18\text{-}29)$$

The matrix representation of $\mathbf{I}_+$ for a single spin i**,** $\mathbf{I}_{i+}$**,** is efficiently expressed in the dimension of the Hilbert

[1] Schiff, pg. 202, equation (27.25).

space of the spin itself, spanning 2($I_i$+1). Examples are shown below for a spin 1/2 and a spin 1 particle

$$\boldsymbol{I}_{i+}\left(I_i = \frac{1}{2}\right) = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \quad \boldsymbol{I}_{i+}(I_i = 1) = \sqrt{2}\begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix} \quad \boldsymbol{I}_{i+}\left(I_i = \frac{3}{2}\right) = \begin{bmatrix} 0 & \sqrt{3} & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & \sqrt{3} \\ 0 & 0 & 0 & 0 \end{bmatrix} \qquad (18\text{-}30)$$

Other useful relationships involving this operator are the following.

$$\boldsymbol{I}_{i+} = (\boldsymbol{I}_{ix} + i\boldsymbol{I}_{iy}) \qquad \boldsymbol{I}_{i+} = (\boldsymbol{I}_{i-})^{\dagger} \qquad (18\text{-}31)$$

$$[\boldsymbol{I}_{i+}, \boldsymbol{I}_{iz}] = -\boldsymbol{I}_{i+}\ , \qquad [\boldsymbol{I}_{i+}, \boldsymbol{I}_{i-}] = 2\boldsymbol{I}_{iz} \qquad (18\text{-}32)$$

and for the spin 1/2 case,

$$\boldsymbol{I}_{i+}\boldsymbol{I}_{i-} = \frac{1}{2} + \boldsymbol{I}_{iz} \qquad \boldsymbol{I}_{i-}\boldsymbol{I}_{i+} = \frac{1}{2} - \boldsymbol{I}_{iz} \qquad (18\text{-}33)$$

It is trivial to verify the properties of the raising operator matrices in equation (18-30). For example,

$$\boldsymbol{I}_{+}|\alpha\rangle = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}\begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} = 0 \qquad \boldsymbol{I}_{+}|\beta\rangle = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}\begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} = |\alpha\rangle \qquad (18\text{-}34)$$

$$\boldsymbol{I}_{+}|\alpha\rangle = \sqrt{2}\begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} = 0\ ,\ \boldsymbol{I}_{+}|\beta\rangle = \sqrt{2}\begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}\begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} = \sqrt{2}\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} = \sqrt{2}|\alpha\rangle\ ,$$

$$\boldsymbol{I}_{+}|\gamma\rangle = \sqrt{2}\begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}\begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} = \sqrt{2}\begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} = \sqrt{2}|\beta\rangle \qquad (18\text{-}35)$$

## 4.4.6 Im

### Usage:

```
#include <HSLib/SpinOp.h>
spin_op Im(spin_sys&, int i);
```

### Description:

The function *Im* is used to produce the lowering operator $\boldsymbol{I}_{-}$ for a single spin, $\boldsymbol{I}_{i-}$ .The result is a spin operator as a matrix in the product (default) basis. The only parameter taken is the spin system.

### Return Value:

The matrix representation of the operator.

### Example(s):

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

```
spin_sys sys(3);
spin_op SOp;
SOp=Im(sys,0);                          // SOp set to I- component for first spin of sys
SOp=Im(sys,2);                          // raising operator for the third spin
}
```

**See Also: Ip, Fm**

**Mathematical Basis:**

The lowering operator $\boldsymbol{I}_-$ is defined to work on state $|I, m\rangle$ as[1]

$$\boldsymbol{I}_-|I, m\rangle = [I(I+1)-m(m-1)]^{1/2}|I, m-1\rangle \quad , \tag{18-36}$$

having matrix elements

$$\langle I, m'|\boldsymbol{I}_-|I, m\rangle = [I(I+1)-m(m-1)]^{1/2}\delta_{m-1, m'} \quad . \tag{18-37}$$

The matrix representation of $\boldsymbol{I}_-$ for a single spin i**,** $\boldsymbol{I}_{i-}$ **,** is efficiently expressed in the dimension of the Hilbert space of the spin itself, spanning 2($I_i$+1). Examples are shown below for particles with spin angular momentum 1/2, 1, and 3/2 respectively.

$$\boldsymbol{I}_{i-}\left(I_i = \frac{1}{2}\right) = \begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix} \quad \boldsymbol{I}_{i-}(I_i = 1) = \sqrt{2}\begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \quad \boldsymbol{I}_{i-}\left(I_i = \frac{3}{2}\right) = \begin{bmatrix} 0 & 0 & 0 & 0 \\ \sqrt{3} & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & \sqrt{3} & 0 \end{bmatrix} \tag{18-38}$$

Other useful relationships involving this operator are the following.

$$\boldsymbol{I}_{i-} = (\boldsymbol{I}_{ix} - \boldsymbol{i}\boldsymbol{I}_{iy}) \qquad \boldsymbol{I}_{i-} = (\boldsymbol{I}_{i+})^{\dagger} \tag{18-39}$$

$$[\boldsymbol{I}_{i-}, \boldsymbol{I}_{iz}] = \boldsymbol{I}_{i-} \ , \quad [\boldsymbol{I}_{i+}, \boldsymbol{I}_{i-}] = 2\boldsymbol{I}_{iz} \tag{18-40}$$

and in the spin 1/2 case,

$$\boldsymbol{I}_{i+}\boldsymbol{I}_{i-} = \frac{1}{2} + \boldsymbol{I}_{iz} \qquad \boldsymbol{I}_{i-}\boldsymbol{I}_{i+} = \frac{1}{2} - \boldsymbol{I}_{iz} \ . \tag{18-41}$$

It is simple verify the properties of the lowering operator matrices in equation (18-38).

$$\boldsymbol{I}_-|\alpha\rangle = \begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix}\begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} = |\beta\rangle \qquad \boldsymbol{I}_-|\beta\rangle = \begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix}\begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} = 0 \tag{18-42}$$

$$\boldsymbol{I}_-|\alpha\rangle = \sqrt{2}\begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} = \sqrt{2}\begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} = \sqrt{2}|\beta\rangle \qquad \boldsymbol{I}_-|\beta\rangle = \sqrt{2}\begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}\begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} = \sqrt{2}\begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} = \sqrt{2}|\gamma\rangle$$

[1] Schiff, pg. 202, equation (27.25).

$$\boldsymbol{I}_{-}|\gamma\rangle = \sqrt{2}\begin{bmatrix}0&0&0\\1&0&0\\0&1&0\end{bmatrix}\begin{bmatrix}0\\0\\1\end{bmatrix} = \begin{bmatrix}0\\0\\0\end{bmatrix} = 0 \qquad (18\text{-}43)$$

### 4.4.7 Ia

**Usage:**

```
#include <HSLib/SpinOp.h>
spin_op Ia (spin_sys &sys, int spin)
```

**Description:**

The function *Ia* provides the polarization operator for a single spin[1], $\boldsymbol{I}_i^{\alpha}$. This function is define exclusively for a spin 1/2 particle. The result is a spin operator as a matrix in the product (default) basis.

**Return Value:**

The matrix representation of the operator in the composite Hilbert space of the spin system.

**Examples:**

```
#include <gamma.h>
main()
  {
  spin_sys ab(2);
  spin_op SOp(ab);
  SOp = Ia(ab, 0);                              // SOp set to polarization operator of first spin of ab.
  }
```

**See Also: Ib, Ipol**

**Mathematical Basis:**

The spin polarization operator $\boldsymbol{I}_i^{\alpha}$ for a single spin 1/2 particle is defined to be

$$\boldsymbol{I}_i^{\alpha} = \frac{1}{2}1 + \boldsymbol{I}_{iz} = \boldsymbol{I}_{i+}\boldsymbol{I}_{i-} \qquad (18\text{-}44)$$

The matrix representation of $\boldsymbol{I}_i^{\alpha}$ is efficiently expressed in the dimension of the Hilbert space of the spin itself, spanning $2(I_i+1)$. This operator is defined exclusively for spins having I = 1/2. The matrix is given below.

$$\boldsymbol{I}_i^{\alpha}\left(I_i = \frac{1}{2}\right) = \begin{bmatrix}1&0\\0&0\end{bmatrix} \qquad (18\text{-}45)$$

For spins with a larger spin quantum number

$$\boldsymbol{I}_i^{\alpha}\left(I_i > \frac{1}{2}\right) = undefined \qquad (18\text{-}46)$$

[1] See Ernst, Bodenhausen, and Wokaun, page 33.

## 4.4.8 Ib

**Usage:**

```
#include <HSLib/SpinOp.h>
spin_op Ib(spin_sys &sys, int spin);
```

**Description:**

The function *Ib* generates the polarization operator[1] for the spin specified, $\boldsymbol{I}_i^{\beta}$. The result is a spin operator as a matrix in the product (default) basis.

**Return Value:**

The matrix representation of the operator in the composite Hilbert space of the spin system.

**Example(s):**

```
#include <gamma.h>
main()
  {
  spin_sys ab(2);
  spin_op SOp(ab);
  SOp=Ib(ab, 1);                              // SOp polarization operator for second spin in ab.
  }
```

**See Also: Ia, Ipol**

**Mathematical Basis:**

The spin polarization operator $\boldsymbol{I}_i^{\beta}$ for a single spin 1/2 particle is defined to be

$$\boldsymbol{I}_i^{\beta} = \frac{1}{2}1 - \boldsymbol{I}_{iz} = \boldsymbol{I}_{i\text{-}}\boldsymbol{I}_{i+} \tag{18-47}$$

The matrix representation of $\boldsymbol{I}_i^{\beta}$ is efficiently expressed in the dimension of the Hilbert space of the spin itself, spanning $2(I_i+1)$. This operator is defined exclusively for spins having I = 1/2. The matrix is given below.

$$\boldsymbol{I}_i^{\beta}\left(I_i = \frac{1}{2}\right) = \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix} \tag{18-48}$$

For spins with a larger spin quantum number

$$\boldsymbol{I}_i^{\beta}\left(I_i > \frac{1}{2}\right) = undefined \tag{18-49}$$

[1] See Ernst, Bodenhausen, and Wokaun, page 33.

## 4.4.9 Ipol

### Usage:

```
#include <HSLib/SpinOp.h>
spin_op Ib(spin_sys &sys, double m, int spin);
```

### Description:

The function *Ipol* generates the polarization operator[1] for the spin specified, $\boldsymbol{I}_i^{\beta}$. The result is a spin operator as a matrix in the product (default) basis.

### Return Value:

The matrix representation of the operator in the composite Hilbert space of the spin system.

### Example(s):

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

### See Also: Ia, Ib

### Mathematical Basis:

The spin polarization operator $\boldsymbol{I}_i^{\beta}$ for a single spin 1/2 particle is defined to be

$$\boldsymbol{I}_i^{\beta} = \frac{1}{2}1 - \boldsymbol{I}_{iz} = \boldsymbol{I}_{i-}\boldsymbol{I}_{i+} \tag{18-50}$$

The matrix representation of $\boldsymbol{I}_i^{\beta}$ is efficiently expressed in the dimension of the Hilbert space of the spin itself, spanning $2(I_i+1)$. This operator is defined exclusively for spins having I = 1/2. The matrix is given below.

$$\boldsymbol{I}_i^{\beta}\left(I_i = \frac{1}{2}\right) = \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix} \tag{18-51}$$

For spins with a larger spin quantum number

$$\boldsymbol{I}_i^{\beta}\left(I_i > \frac{1}{2}\right) = undefined \tag{18-52}$$

[1] See Ernst, Bodenhausen, and Wokaun, page 33.

## 4.5 Multiple-Spin Spin Operators

### 4.5.1 Fe

**Usage:**

```
#include <HSLib/SpinOp.h>
spin_op Fe(spin_sys &sys);
spin_op Fe(spin_sys &sys, int spin);
spin_op Fe(spin_sys &sys, char *iso);
spin_op Fe_sp(spin_sys &sys);
```

**Description:**

The function *Fe* returns the sums of the individual Ie operator components for a whole spin system.

$$\boldsymbol{F}_e = \sum_i^{spins} \boldsymbol{I}_{ie}$$

The result is a spin operator as a matrix in the product (default) basis spanning the full Hilbert space of the input spin system. The only parameter taken is the spin system itself.

The other overloaded version of the function returns the sum of the operator components belonging to the isotopes specified by the second argument, a string. This string must contain the label of the isotope in a format compatible to the notations of class spin_sys.

**Return Value:**

The matrix representation of the operator Fe in the product basis.

**Examples:**

```
#include <gamma.h>
main()
  {
  spin_sys ab(2);
  spin_op SOp;
  SOp=Fe(ab);                              //
  }
```

**See Also: Ie, Fx, Fy, Fz**

## 4.5.2 Fx

### Usage:

```
#include <HSLib/SpinOp.h>
spin_op Fx(spin_sys &sys);
spin_op Fx(spin_sys &sys, int spin);
spin_op Fx(spin_sys &sys, char *iso);
spin_op Fx_sp(spin_sys &sys);
```

### Description:

Function *Fx* returns a sum over individual Ix operator components for a set of specified spins in a spin system.

$$\boldsymbol{F}_x = \sum_{\{i\}}^{spins} \boldsymbol{I}_{ix}$$

1. Fx(spin_sys &sys) - Returns the operator containing all spin components for the entire spin system.
2. Fx(spin_sys &sys, int spin) - Operator containing only a component from the spin specified. Equal to Ix.
3. Fx(spin_sys &sys, char *iso) - Operator containing all spin components of the isotope type specified.

The resulting spin operator (matrix) is in the product basis spanning the full Hilbert space of the spin system.

### Return Value:

A spin operator with matrix representation in the composite Hilbert space of the spin system.

### Example(s):

```
#include <gamma.h>
main()
  {
  spin_sys CH3(4);                      // define a four spin system, CH3.
  CH3.isotope(0, “13C”);                // set the first spin to a carbon 13, rest 1H by default.
  spin_op SOp;                          // define a spin operator SOp
  SOp = Fx(CH3);                        // set spin operator SOp equal to Fx of the spin system
  SOp = Fx(CH3,0);                      // reset SOp to Fx spin zero = Ix spin zero
  SOp = Fx(CH3, “1H”);                  // reset spin operator SOp equal to Fx for protons.
  }
```

### See Also: Fy, Fz, Ix, Iy, Iz.

### Mathematical Basis:

Consider a two spin system composed of a spin 1 and a spin 1/2 particle. The matrix for ***Fx*** is obtained from a sum over both spin components and by necessity in the composite Hilbert space of the spin system

$$\boldsymbol{F}_x = \boldsymbol{I}_{1x} + \boldsymbol{I}_{2x} = \frac{1}{\sqrt{2}}\begin{bmatrix} 0&0&1&0&0&0\\ 0&0&0&1&0&0\\ 1&0&0&0&1&0\\ 0&1&0&0&0&1\\ 0&0&1&0&0&0\\ 0&0&0&1&0&0 \end{bmatrix} + \frac{1}{2}\begin{bmatrix} 0&1&0&0&0&0\\ 1&0&1&0&0&0\\ 0&1&0&1&0&0\\ 0&0&1&0&1&0\\ 0&0&0&1&0&1\\ 0&0&0&0&1&0 \end{bmatrix} = \frac{1}{2}\begin{bmatrix} 0&1&\sqrt{2}&0&0&0\\ 1&0&1&\sqrt{2}&0&0\\ \sqrt{2}&1&0&1&\sqrt{2}&0\\ 0&\sqrt{2}&1&0&1&\sqrt{2}\\ 0&0&\sqrt{2}&1&0&1\\ 0&0&0&\sqrt{2}&1&0 \end{bmatrix}.$$

## 4.5.3 Fy

### Usage:

```
#include <HSLib/SpinOp.h>
spin_op Fy(spin_sys &sys);
spin_op Fy(spin_sys &sys, int spin);
spin_op Fy(spin_sys &sys, char *iso);
spin_op Fy_sp(spin_sys &sys);
```

### Description:

Function *Fy* returns a sum over individual Iy operator components for a set of specified spins in a spin system.

$$\boldsymbol{F}_y = \sum_{\{i\}}^{spins} \boldsymbol{I}_{iy}$$

1. Fy(spin_sys &sys) - Returns the operator containing all spin components for the entire spin system.
2. Fy(spin_sys &sys, int spin) - Operator with only the component from the spin specified. Equivalent to Iy.
3. Fy(spin_sys &sys, char *iso) - Operator containing all spin components of the isotope type specified.

Result is a spin operator as a matrix in the product (default) basis which spans the full Hilbert space of the spin system.

### Return Value:

A spin operator with matrix representation in the composite Hilbert space of the spin system.

### Example(s):

```
#include <gamma.h>
main()
  {
  spin_sys thiophene(4);                   // define 4 spin system, thiophene. All 1H (default).
  spin_op SOp;                             // define a spin operator SOp
  SOp = Fy(thiophene);                     // set spin operator SOp equal to Fy of the spin system
  SOp = Fy(thiophene, 1);                  // reset SOp equal to Fy of spin two = Iy spin two
  SOp = Fy(thiophene, “1H”);               // reset SOp equal to Fy for protons, here = case 1.
  }
```

### See Also: Fx, Fz, Ix, Iy, Iz.

### Mathematical Basis:

Consider a two spin system composed of spin 1/2 particles. The matrix for $\boldsymbol{F}_y$ is obtained from a sum over both spin components and, by necessity, in the composite Hilbert space of the spin system. From equations (18-15) and (18-16) we have the following matrix.

$$\boldsymbol{F}_y = \boldsymbol{I}_{1y} + \boldsymbol{I}_{2y} = \frac{i}{2}\begin{bmatrix} 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & -1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} + \frac{i}{2}\begin{bmatrix} 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 \\ 0 & 0 & 1 & 0 \end{bmatrix} = \frac{i}{2}\begin{bmatrix} 0 & -1 & -1 & 0 \\ 1 & 0 & 0 & -1 \\ 1 & 0 & 0 & -1 \\ 0 & 1 & 1 & 0 \end{bmatrix} \qquad (18\text{-}53)$$

## 4.5.4 Fz

### Usage:

```
#include <HSLib/SpinOp.h>
spin_op Fz(spin_sys &sys);
spin_op Fz(spin_sys &sys, int spin);
spin_op Fz(spin_sys &sys, char *iso);
spin_op Fz_sp(spin_sys &sys);
```

### Description:

Function *Fz* sums of the individual Iz operator components for a set of specified spins in a spin system.

$$\boldsymbol{F}_z = \sum_{\{i\}}^{spins} \boldsymbol{I}_{iz}$$

1. Fz(spin_sys &sys) - Returns the operator containing all spin components for the entire spin system.
2. Fz(spin_sys &sys, int spin) - Operator containing only a component from the spin specified. Equal to Iz.
3. Fz(spin_sys &sys, char *iso) - Operator containing all spin components of the isotope type specified.

Result spin operator (matrix) is in the product basis spanning the full Hilbert space of the spin system.

### Return Value:

A spin operator with matrix representation in the composite Hilbert space of the spin system.

### Examples:

```
#include <gamma.h>
main()
  {
  spin_op SOp;                          // define a spin operator SOp
  spin_sys CH3(3);                      // define a spin system, 3 spins, CH3
  CH3.isotope(0, “13C”);                // set spin 1 to a carbon 13, rest are 1H by default.
  SOp = Fz(CH3);                        // set spin operator SOp equal to Fz of the spin system
  SOp = Fz(CH3,0);                      // reset SOp equal to Fz of spin zero = Iz spin zero
  SOp = Fz(CH3, “1H”);                  // reset spin operator SOp equal to Fz for protons.
  }
```

### See Also: Fx, Fy, Ix, Iy, Iz.

### Mathematical Basis:

For example, consider a two spin system composed of a spin 1 and a spin 1/2 particle. The matrix for $\boldsymbol{F}_z$ is obtained from a sum over both spin components and, by necessity, in the composite Hilbert space of the spin system. From equations (18-23) and (18-24) we have the following matrix.

$$\boldsymbol{F}_z = \boldsymbol{I}_{1z} + \boldsymbol{I}_{2z} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 \end{bmatrix} + \frac{1}{2}\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 \end{bmatrix} = \frac{1}{2}\begin{bmatrix} 3 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & -3 \end{bmatrix} \qquad (18\text{-}54)$$

## 4.5.5 Fp

### Usage:

```
#include <HSLib/SpinOp.h>
spin_op Fp(spin_sys &sys);
spin_op Fp(spin_sys &sys, int spin);
spin_op Fp(spin_sys &sys, char *iso);
spin_op Fp_sp(spin_sys &sys);
```

### Description:

Function *Fp* returns a sum over individual Ip operator components for a set of specified spins in a spin system.

$$\boldsymbol{F}_p = \sum_{\{i\}}^{spins} \boldsymbol{I}_{ip} = \sum_{\{i\}}^{spins} \boldsymbol{I}_{i+}$$

1. Fp(spin_sys &sys) - Returns the operator containing all spin components for the entire spin system.
2. Fp(spin_sys &sys, int spin) - Operator with only the component from the spin specified. Equivalent to Ip.
3. Fp(spin_sys &sys, char *iso) - Operator containing all spin components of the isotope type specified.

Result spin operator (matrix) is in the product basis which spans the full Hilbert space of the spin system.

### Return Value:

A spin operator with matrix representation in the composite Hilbert space of the spin system.

### Example(s):

```
#include <gamma.h>
main()
  {
  spin_sys thiophene(4);              // 4 spin system, thiophene. All spins default 1H.
  spin_op SOp(thiophene);             // spin operator SOp associated with the spin system
  SOp = Fp(thiophene);                // set spin operator SOp equal to Fp of the spin system
  SOp = Fp(thiophene, 1);             // reset SOp equal to Fp of spin two = Iy spin two
  SOp = Fp(thiophene, “1H”);          // reset SOp to Fp for protons, here equal to case 1.
  }
```

**See Also: Fm, Ip, Im.**

## 4.5.6 Fm

### Usage:

```
#include <HSLib/SpinOp.h>
spin_op Fm(spin_sys &sys);
spin_op Fm(spin_sys &sys, int spin);
spin_op Fm(spin_sys &sys, char *iso);
spin_op Fm_sp(spin_sys &sys);
```

### Description:

***Fm*** returns a sum over individual $\boldsymbol{I}_-$ operator components for a set of specified spins in a spin system.

$$\boldsymbol{F}_m = \sum_i^{spins} \boldsymbol{I}_{im} = \sum_i^{spins} \boldsymbol{I}_{i-}$$

1. Fm(spin_sys &sys) - Returns the operator containing all spin components for the entire spin system.
2. Fm(spin_sys &sys, int spin) - Operator with only the component from a spin specified. Equivalent to Im.
3. Fm(spin_sys &sys, char *iso) - Operator containing all spin components of the isotope type specified.

Result spin operator (matrix) is in the product basis which spans the full Hilbert space of the spin system.

### Return Value:

A spin operator with matrix representation in the composite Hilbert space of the spin system.

### Example(s):

```
#include <gamma.h>
main()
  {
  CH3.isotope(0, “13C”);                    // set first spin to a carbon 13, rest are 1H by default.
  spin_op SOp;                              // define a spin operator SOp
  SOp = Fm(CH3);                            // set spin operator SOp to Fm of the spin system
  SOp = Fm(CH3,0);                          // reset SOp equal to Fm of spin zero = Im spin zero
  SOp = Fm(CH3, “1H”);                      // reset spin operator SOp equal to Fm for protons.
  }
```

**See Also: Fp, Ip, Im.**

## 4.5.7 Fa

### Usage:

```
#include <HSLib/SpinOp.h>
spin_op Fa(spin_sys &sys);
spin_op Fa(spin_sys &sys, int spin);
spin_op Fa(spin_sys &sys, char *iso);
spin_op Fa_sp(spin_sys &sys);
```

### Description:

The function *Fa* returns the sums of the individual Ia operator components for a set of specified spins in a spin system.

$$\boldsymbol{F}_a = \sum_{\{i\}}^{spins} \boldsymbol{I}_{ia} = \sum_{\{i\}}^{spins} \boldsymbol{I}_i^{\alpha}$$

1. Fa(spin_sys &sys) - Returns the operator containing all spin components for the entire spin system.
2. Fa(spin_sys &sys, int spin) - Returns the operator containing only the component from the spin specified. Equivalent to Ia.
3. Fa(spin_sys &sys, char *iso) - Returns the operator containing all spin components of the isotope type specified.

The result is a spin operator as a matrix in the product (default) basis which spans the full Hilbert space of the spin system.

### Return Value:

A spin operator with matrix representation in the composite Hilbert space of the spin system.

### Example(s):

```
#include <gamma.h>
main()
  {
  CH3.isotope(0, “13C”);                  // set first spin to a carbon 13, rest are 1H by default.
  spin_op SOp;                            // define a spin operator SOp
  SOp = Fz(CH3);                          // set spin operator SOp equal to Fz of the spin system
  SOp = Fz(CH3,0);                        // reset SOp equal to Fz of spin zero = Iz spin zero
  SOp = Fz(CH3, “1H”);                    // reset spin operator SOp equal to Fz for protons.
  }
```

**See Also: Fb, Ia, Ib.**

## 4.5.8 Fb

### Usage:

```
#include <HSLib/SpinOp.h>
spin_op Fb(spin_sys &sys);
spin_op Fb(spin_sys &sys, int spin);
spin_op Fb(spin_sys &sys, char *iso);
spin_op Fb_sp(spin_sys &sys);
```

### Description:

The function *Fb* returns the sums of the individual Ib operator components for a set of specified spins in a spin system.

$$\boldsymbol{F}_b = \sum_{\{i\}}^{spins} \boldsymbol{I}_{ib} = \sum_{\{i\}}^{spins} \boldsymbol{I}_i^{\beta}$$

1. Fb(spin_sys &sys) - Returns the operator containing all spin components for the entire spin system.
2. Fb(spin_sys &sys, int spin) - Returns the operator containing only the component from the spin specified. Equivalent to Ib.
3. Fb(spin_sys &sys, char *iso) - Returns the operator containing all spin components of the isotope type specified.

The result is a spin operator as a matrix in the product (default) basis which spans the full Hilbert space of the spin system.

### Return Value:

A spin operator with matrix representation in the composite Hilbert space of the spin system.

### Example(s):

```
#include <gamma.h>
main()
  {
  CH3.isotope(0, “13C”);                  // set first spin to a carbon 13, rest are 1H by default.
  spin_op SOp;                            // define a spin operator SOp
  SOp = Fb(CH3);                          // set spin operator SOp equal to Fb of the spin system
  SOp = Fb(CH3,0);                        // reset SOp equal to Fb of spin zero = Ib spin zero
  SOp = Fb(CH3, “1H”);                    // reset spin operator SOp equal to Fb for protons.
  }
```

**See Also: Fa, Ia, Ib.**

## 4.5.9 Fpol

### Usage:

```
#include <HSLib/SpinOp.h>
spin_op Fpol(spin_sys &sys, double m);
spin_op Fpol(spin_sys &sys, double m, int spin);
spin_op Fpol(spin_sys &sys, double m, char *iso);
spin_op Fpol_sp(spin_sys &sys, double m);
```

### Description:

The function *Fb* returns the sums of the individual Ib operator components for a set of specified spins in a spin system.

$$\boldsymbol{F}_b = \sum_{\{i\}}^{spins} \boldsymbol{I}_{ib} = \sum_{\{i\}}^{spins} \boldsymbol{I}_i^{\beta} \qquad (18\text{-}55)$$

1. Fb(spin_sys &sys) - Returns the operator containing all spin components for the entire spin system.
2. Fb(spin_sys &sys, int spin) - Returns the operator containing only the component from the spin specified. Equivalent to Ib.
3. Fb(spin_sys &sys, char *iso) - Returns the operator containing all spin components of the isotope type specified.

The result is a spin operator as a matrix in the product (default) basis which spans the full Hilbert space of the spin system.

### Return Value:

A spin operator with matrix representation in the composite Hilbert space of the spin system.

### Example(s):

```
#include <gamma.h>
main()
  {
  CH3.isotope(0, “13C”);                  // set first spin to a carbon 13, rest are 1H by default.
  spin_op SOp;                            // define a spin operator SOp
  SOp = Fb(CH3);                          // set spin operator SOp equal to Fb of the spin system
  SOp = Fb(CH3,0);                        // reset SOp equal to Fb of spin zero = Ib spin zero
  SOp = Fb(CH3, “1H”);                    // reset spin operator SOp equal to Fb for protons.
  }
```

**See Also: Fa, Ia, Ib.**

## 4.5.10 Fpdt

### Usage:

```
#include <HSLib/SpinOp.h>
spin_op Fpdt (spin_sys &sys, String pdt);
```

### Description:

The function *Fpdt* returns a spin operator which is a product of single spin operators.

$$\boldsymbol{F}_{pdt} = \prod_{i}^{spins} \boldsymbol{I}_{i\kappa} \qquad \kappa = \{x, y, z, p, m, e, 0, 1, 2, 3, +,'-,a,b\}$$

The single spin operators are specified by the string pdt. Each letter of the string dictates which single spin operator to use in the product, the first letter for the first spin, the second for the second spin and so on. Valid letters are the following

| Letter:Operator | Letter:Operator |
|---|---|
| e:identity matrix | 0:identity matrix |
| x:Ix | 1:Ix |
| y:Iy | 2:Iy |
| z:Iz | 3:Iz |
| p:I+ | +:I+ |
| m:I- | -:I- |
| a:Ialpha | b:Ibeta |

The result is a spin operator as a matrix in the product (default) basis which spans the full Hilbert space of the spin system.

## Return Value:

A spin operator with matrix representation in the composite Hilbert space of the spin system.

## Example(s):

```
#include <gamma.h>
main()
  {
  spin_sys ab(2);                          // 2 spin system, ab (defaults to both I = 1/2 spins).
  spin_sys abc(3);                         // 3 spin system, abc (defaults to all I = 1/2 spins).spin_op SOp;
                                       // define a spin operator SOp
  SOp = Fpdt(ab,"+-");                     // set spin operator SOp equal to I+(0)I-(1)
  SOp = Fpdt(ab,"0z");                     // set spin operator SOp equal to Iz(1)
  SOp = Fpdt(ab,"p0z");                    // set spin operator SOp equal to I+(0), z is ignored.
  SOp = Fpdt(abc,"p0z");                   // set spin operator SOp equal to I+(0)Iz(2).
  }
```

## See Also:

# 4.6 Rotation Operators

## 4.6.1 Rx

### Usage:

```
#include <HSLib/SpinOp.h>
spin_op Rx (spin_sys &sys, double beta);
spin_op Rx (spin_sys &sys, int spin, double beta);
spin_op Rx (spin_sys &sys, char *iso, double beta);
spin_op Rx_sp (spin_sys &sys, double beta);
```

### Description:

The function *Rx* returns the spin rotation operator, $\mathbf{R}_x(\beta)$, for rotating spin angular momentum about the x-axis by the angle $\beta$. This is defined by the formula

$$\boldsymbol{R}_x(\beta) = exp(-i\beta\boldsymbol{J}_x) \tag{18-56}$$

where $\mathbf{J}_x$ is an appropriate spin angular momentum operator.

1. Rx(spin_sys &sys, double beta) - Returns the rotation matrix which rotates the entire spin system by the angle beta about the x-axis. Beta is specified in degrees.

$$\boldsymbol{R}_x(\beta) = exp(-i\beta\boldsymbol{F}_x) \tag{18-57}$$

2. Rx(spin_sys &sys, int spin, double beta) - Returns the rotation matrix which rotates the spin specified by the angle beta about the x-axis. Beta is specified in degrees.

$$\boldsymbol{R}_{ix}(\beta) = exp(-i\beta\boldsymbol{I}_{ix}) \tag{18-58}$$

3. Rx(spin_sys &sys, char *iso, double beta) - Returns the rotation matrix which rotates the spins of the isotope type specified by the angle beta about the x-axis. Beta is specified in degrees.

$$\boldsymbol{R}_{\{i\}x}(\beta) = exp(-i\beta\boldsymbol{F}_{\{i\}x}) \quad \text{where } j = \{i\}\forall \quad \text{spins } j \ni \gamma_j = \gamma_i \tag{18-59}$$

### Return Value:

The matrix representation of the spin rotation operator in the composite Hilbert space of the spin system.

### Example(s):

```
#include <gamma.h>
main()
  {
  spin_sys ab(2);                          // two spin system, ab (defaults to both I = 1/2 spins).
  spin_op SOp;                             // define a spin operator SOp
  SOp = Rx(ab,90.0);                       // set spin operator SOp equal to Rx(90)
```

}

**See Also: Ry, Rz, Rxy.**

**Mathematical Basis:**

All rotation operators provided by the function Rx are computed from single spin rotation operators $\boldsymbol{R}_{ix}(\beta)$ by taking the appropriate direct products in accordance with equation (18-106). The formulas used to determine individual $\boldsymbol{R}_{ix}(\beta)$ matrices are obtained from equation (18-123) in the case of a spin 1/2 particle,

$$\boldsymbol{R}_{ix}^{(1/2)}(\beta) = 1\cos(\beta/2) - i2\boldsymbol{I}_{ix}\sin(\beta/2) \quad , \tag{18-60}$$

and from the general equation (18-111) in the case of a spin with I > 1/2,

$$\boldsymbol{R}_{ix}(\beta) = exp(-i\beta\boldsymbol{I}_{ix}) \ . \tag{18-61}$$

Substitution of equation (18-8) into equation (18-60) yields the explicit rotation matrix for this single spin 1/2 treatment.[1]

$$\boldsymbol{R}_{ix}^{(1/2)}(\beta) = \begin{bmatrix} \cos(\beta/2) & -i\sin(\beta/2) \\ -i\sin(\beta/2) & \cos(\beta/2) \end{bmatrix} \tag{18-62}$$

Unfortunately, there is currently no similar simple formula for a spin with I > 1/2 and these are computed by actual exponentiation of the operator as stated in (18-61). A few specific rotation matrices one will obtain from this function are as follows.

$$\boldsymbol{R}_{ix}^{(1/2)}\left(\frac{\pi}{3}\right) = \begin{bmatrix} 0.866 & -0.5i \\ -0.5i & 0.866 \end{bmatrix} \qquad \boldsymbol{R}_{ix}^{(1/2)}\left(\frac{\pi}{2}\right) = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 & -i \\ -i & 1 \end{bmatrix} \qquad \boldsymbol{R}_{ix}^{(1/2)}(\pi) = \begin{bmatrix} 0 & -i \\ -i & 0 \end{bmatrix} \tag{18-63}$$

$$\boldsymbol{R}_{ix}^{(1)}\left(\frac{\pi}{3}\right) = \frac{1}{4}\begin{bmatrix} 3 & -2.45i & -1 \\ -2.45i & 2 & -2.45i \\ -1 & -2.45i & 3 \end{bmatrix} \qquad \boldsymbol{R}_{ix}^{(1)}\left(\frac{\pi}{2}\right) = \frac{1}{2}\begin{bmatrix} 1 & -\sqrt{2}i & -1 \\ -\sqrt{2}i & 0 & -\sqrt{2}i \\ -1 & -\sqrt{2}i & 1 \end{bmatrix} \qquad \boldsymbol{R}_{ix}^{(1)}(\pi) = \frac{1}{2}\begin{bmatrix} 0 & 0 & -1 \\ 0 & -1 & 0 \\ -1 & 0 & 0 \end{bmatrix} \tag{18-64}$$

$$\boldsymbol{R}_{ix}^{(3/2)}\left(\frac{\pi}{2}\right) = \begin{vmatrix} 0.35 & -0.61i & -0.61 & 0.35i \\ -0.61i & -0.35 & -0.35i & -0.61 \\ -0.61 & -0.35i & -0.35 & -0.61i \\ 0.35i & -0.61 & -0.61i & 0.35 \end{vmatrix} \qquad \boldsymbol{R}_{ix}^{(3/2)}(\pi) = \begin{vmatrix} 0 & 0 & 0 & i \\ 0 & 0 & i & 0 \\ 0 & i & 0 & 0 \\ i & 0 & 0 & 0 \end{vmatrix} \tag{18-65}$$

For a multiple spin system one may obtain the rotation matrix in the composite Hilbert space by taking direct products of the single spin operators in accordance with equation (18-142). As an example, if we consider a two spin system composed of a spin 1/2 and a spin 1 particle we have the following matrices.

[1] See Ernst, Bodenhausen, and Wokaun, page 405, equation (8.1.4).

$$\boldsymbol{R}_{1x}(\pi) \;=\; \boldsymbol{R}_{1x}(\pi)\otimes\boldsymbol{I}_{2e} \;=\; \begin{bmatrix} 0 & -i \\ -i & 0 \end{bmatrix} \otimes \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \;=\; -i\begin{vmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{vmatrix} \tag{18-66}$$

$$\boldsymbol{R}_{2x}(\pi) \;=\; \boldsymbol{I}_{1e}\otimes\boldsymbol{R}_{2x}(\pi) \;=\; \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \otimes \frac{1}{2}\begin{bmatrix} 0 & 0 & -1 \\ 0 & -1 & 0 \\ -1 & 0 & 0 \end{bmatrix} \;=\; -1\begin{vmatrix} 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{vmatrix} \tag{18-67}$$

$$\boldsymbol{R}_{1x}(\pi) \;=\; \boldsymbol{R}_{1x}(\pi)+\boldsymbol{R}_{2x}(\pi) \;=\; -1\begin{vmatrix} 0 & 0 & 1 & i & 0 & 0 \\ 0 & 1 & 0 & 0 & i & 0 \\ 1 & 0 & 0 & 0 & 0 & i \\ i & 0 & 0 & 0 & 0 & 1 \\ 0 & i & 0 & 0 & 1 & 0 \\ 0 & 0 & i & 1 & 0 & 0 \end{vmatrix} \tag{18-68}$$

Finally, we consider the effect of these rotations on the state vectors of a spin 1/2 particle. It we use

$$|\alpha\rangle \;=\; \begin{bmatrix} 1 \\ 0 \end{bmatrix} \qquad \text{and} \qquad |\beta\rangle \;=\; \begin{bmatrix} 0 \\ 1 \end{bmatrix} \tag{18-69}$$

Then a rotation by β degrees about x has the following effect.

$$[\boldsymbol{R}_x^{(1/2)}(\beta)]|\alpha\rangle \;=\; \begin{bmatrix} \cos(\beta/2) & -i\sin(\beta/2) \\ -i\sin(\beta/2) & \cos(\beta/2) \end{bmatrix}\begin{bmatrix} 1 \\ 0 \end{bmatrix} \;=\; \begin{bmatrix} \cos(\beta/2) \\ -i\sin(\beta/2) \end{bmatrix} \tag{18-70}$$

$$[\boldsymbol{R}_x^{(1/2)}(\beta)]|\beta\rangle \;=\; \begin{bmatrix} \cos(\beta/2) & -i\sin(\beta/2) \\ -i\sin(\beta/2) & \cos(\beta/2) \end{bmatrix}\begin{bmatrix} 0 \\ 1 \end{bmatrix} \;=\; \begin{bmatrix} -i\sin(\beta/2) \\ \cos(\beta/2) \end{bmatrix} \tag{18-71}$$

Then a rotation by β degrees about x has the following effect.

$$\left[\boldsymbol{R}_x^{(1/2)}\left(\frac{\pi}{2}\right)\right]|\alpha\rangle \;=\; \frac{1}{\sqrt{2}}\begin{bmatrix} 1 \\ -i \end{bmatrix} \qquad [\boldsymbol{R}_x^{(1/2)}(\pi)]|\alpha\rangle \;=\; \begin{bmatrix} 0 \\ -i \end{bmatrix} \tag{18-72}$$

$$\left[\boldsymbol{R}_x^{(1/2)}\left(\frac{\pi}{2}\right)\right]|\beta\rangle \;=\; \frac{1}{\sqrt{2}}\begin{bmatrix} -i \\ 1 \end{bmatrix} \qquad [\boldsymbol{R}_x^{(1/2)}(\pi)]|\beta\rangle \;=\; \begin{bmatrix} -i \\ 0 \end{bmatrix} \tag{18-73}$$

## 4.6.2 Ry

### Usage:

```
#include <HSLib/SpinOp.h>
spin_op Ry (spin_sys &sys, double beta);
spin_op Ry (spin_sys &sys, int spin, double beta);
spin_op Ry (spin_sys &sys, char *iso, double beta);
spin_op Ry_sp (spin_sys &sys, double beta);
```

### Description:

The function *Ry* returns the spin rotation operator, $\boldsymbol{R}_y(\beta)$ , for rotating spin angular momentum about the y-axis by the angle β. This is defined by the formula

$$\boldsymbol{R}_y(\beta) = exp(-i\beta\boldsymbol{J}_y) \tag{18-74}$$

where $\boldsymbol{J}_y$ is an appropriate spin angular momentum operator.

1. Ry(spin_sys &sys, double beta) - Returns the rotation matrix which rotates the entire spin system by the angle beta about the y-axis. Beta is specified in degrees.

$$\boldsymbol{R}_y(\beta) = exp(-i\beta\boldsymbol{F}_y) \tag{18-75}$$

2. Ry(spin_sys &sys, int spin, double beta) - Returns the rotation matrix which rotates the spin specified by the angle beta about the y-axis. Beta is specified in degrees.

$$\boldsymbol{R}_{iy}(\beta) = exp(-i\beta\boldsymbol{I}_{iy}) \tag{18-76}$$

3. Ry(spin_sys &sys, char *iso, double beta) - Returns the rotation matrix which rotates the spins of the isotope type specified by the angle beta about the y-axis. Beta is specified in degrees.

$$\boldsymbol{R}_{\{i\}y}(\beta) = exp(-i\beta\boldsymbol{F}_{\{iy\}}) \quad \text{where } j = \{i\}\forall \quad \text{spins } j \ni \gamma_j = \gamma_i \tag{18-77}$$

### Return Value:

The matrix representation of the spin rotation operator Ry.

### Example(s):

```
#include <gamma.h>
main()
  {
  spin_sys ab(2);                    // define a two spin system, ab (default both I = 1/2 ).
  spin_op SOp(ab);                   // spin operator SOp associated with the spin system
  SOp = Ry(ab);                      // set spin operator SOp to Ry of the spin system
  }
```

### See Also: Rx, Rz, Rxy

### Mathematical Basis:

All rotation operators provided by the function Ry are computed from single spin rotation operators $\boldsymbol{R}_{iy}(\beta)$ by taking the appropriate direct products in accordance with equation (18-106). The formulas used to determine individual $\boldsymbol{R}_{iy}(\beta)$ matrices are obtained from equation (18-123) in the case of a spin 1/2 particle,

$$\boldsymbol{R}_y(\beta) = 1\cos(\beta/2) - i2\boldsymbol{I}_y\sin(\beta/2) \quad , \tag{18-78}$$

and from equation (18-110) in the case of a spin with I > 1/2,

$$\boldsymbol{R}_{iy}(\beta) = exp(-i\beta\boldsymbol{I}_{iy}) \ . \tag{18-79}$$

Substitution of equation (18-14) into equation (18-60) yields the explicit rotation matrix for this single spin 1/2 treatment.

$$\boldsymbol{R}_y(\beta) = \begin{bmatrix} \cos(\beta/2) & -\sin(\beta/2) \\ \sin(\beta/2) & \cos(\beta/2) \end{bmatrix} \quad . \tag{18-80}$$

Unfortunately, there is currently no similar simple formula for a spin with I > 1/2 and these are computed by actual exponentiation of the operator as stated in (18-61). A few specific rotation matrices one will obtain from this function are as follows.

$$\boldsymbol{R}_{iy}^{(1/2)}\left(\frac{\pi}{3}\right) = \begin{bmatrix} 0.866 & -0.5 \\ 0.5 & 0.866 \end{bmatrix} \qquad \boldsymbol{R}_{iy}^{(1/2)}\left(\frac{\pi}{2}\right) = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 & -1 \\ 1 & 1 \end{bmatrix} \qquad \boldsymbol{R}_{iy}^{(1/2)}(\pi) = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \tag{18-81}$$

One may obtain the rotation matrix in a multiple spin system by taking direct products of the single spin operators in accordance with equation (18-142). This particular rotation corresponds directly to the reduced Wigner rotation matrices[1]

$$\boldsymbol{R}_{iy}^{(1/2)}(\beta) = d^{(1/2)}(\beta) \quad . \tag{18-82}$$

## 4.6.3 Rz

### Usage:

```
#include <HSLib/SpinOp.h>
spin_op Rz (spin_sys &sys, double beta);
spin_op Rz (spin_sys &sys, int spin, double beta);
spin_op Rz (spin_sys &sys, char *iso, double beta);
spin_op Rz_sp (spin_sys &sys, double beta);
```

### Description:

The function *Rz* returns the spin rotation operator, $\boldsymbol{R}_z(\beta)$ , for rotating spin angular momentum about the z-axis by the angle β. This is defined by the formula

$$\boldsymbol{R}_z(\beta) = exp(-i\beta\boldsymbol{J}_z) \tag{18-83}$$

where $\boldsymbol{J}_z$ is an appropriate spin angular momentum operator.

1. Rz(spin_sys &sys, double beta) - Returns the rotation matrix which rotates the entire spin system by the

[1] See Brink and Satchler, page 25.

angle beta about the z-axis. Beta is specified in degrees.

$$\boldsymbol{R}_z(\beta) = exp(-i\beta\boldsymbol{F}_z) \tag{18-84}$$

2. Rz(spin_sys &sys, int spin, double beta) - Returns the rotation matrix which rotates the spin specified by the angle beta about the z-axis. Beta is specified in degrees.

$$\boldsymbol{R}_{iz}(\beta) = exp(-i\beta\boldsymbol{I}_{iz}) \tag{18-85}$$

3. Rz(spin_sys &sys, char *iso, double beta) - Returns the rotation matrix which rotates the spins of the isotope type specified by the angle beta about the z-axis. Beta is specified in degrees.

$$\boldsymbol{R}_{\{i\}z}(\beta) = exp(-i\beta\boldsymbol{F}_{\{i\}z}) \quad \text{where } j = \{i\}\forall \quad \text{spins } j \ni \gamma_j = \gamma_i \tag{18-86}$$

### Return Value:

The matrix representation of the spin rotation operator Rz.

### Example(s):

```
#include <gamma.h>
main()
  {
  spin_sys ab(2);                          // 2 spin system, ab (defaults to both I = 1/2 particles).
  spin_op SOp(ab);                         // spin operator SOp associated with the spin system
  SOp = Rz(ab, 90.0);                      // set SOp equal to Rz(PI/2) of the spin system
  }
```

### See Also: Rx, Ry, Ryz, Rzx

### Mathematical Basis:

All rotation operators provided by the function Rz are computed from single spin rotation operators $\boldsymbol{R}_{iz}(\beta)$ by taking the appropriate direct products in accordance with equation (18-106). The formulas used to determine individual $\boldsymbol{R}_{iz}(\beta)$ matrices are obtained from equation (18-123) in the case of a spin 1/2 particle,

$$\boldsymbol{R}_z(\beta) = 1\cos(\beta/2) - i2\boldsymbol{I}_z\sin(\beta/2) \quad , \tag{18-87}$$

and from equation (18-110) in the case of a spin with I > 1/2,

$$\boldsymbol{R}_{iz}(\beta) = exp(-i\beta\boldsymbol{I}_{iz}) \ . \tag{18-88}$$

Substitution of equation (18-22) into equation (18-60) yields the explicit rotation matrix for this single spin 1/2 treatment.[1]

$$\boldsymbol{R}_z(\beta) = \begin{bmatrix} exp[-i(\beta/2)] & 0 \\ 0 & exp[i(\beta/2)] \end{bmatrix} \quad , \tag{18-89}$$

and similarly, by blending equation (18-22) with equation (18-89)for a single spin 1 particle the matrix representation of $\mathbf{R}_z(\beta)$ will look like

[1] See Goldman, page 59, equation (3.36).

$$\boldsymbol{R}_z(\beta) = \begin{bmatrix} exp(-i\beta) & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & exp(i\beta) \end{bmatrix} . \tag{18-90}$$

Unfortunately, there is currently no simple general formula for a spin with arbitrary I and these are computed by actual exponentiation of the operator as stated in (18-61). A few specific rotation matrices one will obtain from this function are as follows.

$$\boldsymbol{R}_{iz}^{(1/2)}\left(\frac{\pi}{3}\right) = \begin{bmatrix} 0.866 - 0.5i & 0 \\ 0 & 0.866 + 0.5i \end{bmatrix} \qquad \boldsymbol{R}_{iz}^{(1/2)}\left(\frac{\pi}{2}\right) = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 + i & 0 \\ 0 & 1 + i \end{bmatrix} \tag{18-91}$$

$$\boldsymbol{R}_{iz}^{(1/2)}(\pi) = \begin{bmatrix} -i & 0 \\ 0 & i \end{bmatrix} \tag{18-92}$$

One may obtain the rotation matrix in a multiple spin system by taking direct products of the single spin operators in accordance with equation (18-142).

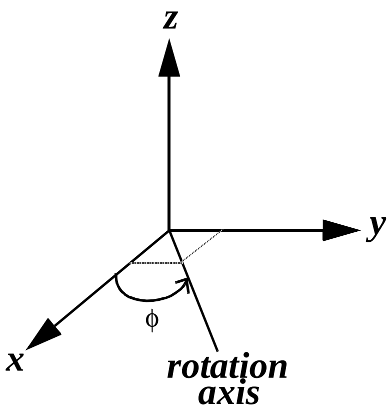
### 4.6.4 Rxy

**Usage:**

```
#include <HSLib/SpinOp.h>
spin_op Rxy (spin_sys &sys, double phi, double beta);
spin_op Rxy (spin_sys &sys, int spin, double phi, double beta);
spin_op Rxy (spin_sys &sys, char *iso, double phi, double beta);
spin_op Rxy_sp (spin_sys &sys, double phi, double beta);
```

**Description:**

The function *Rxy* returns the spin rotation operator $\mathbf{R}_{xy}(\phi,\beta)$ which rotates spin angular momentum by an angle β about an axis in the xy-plane which is φ degrees over from the x-axis. The angle β is the rotation angle, and the angle φ the phase angle as shown in the figure below.



When the phase angle φ is zero, $\boldsymbol{R}_{xy}(0, \beta) = \boldsymbol{R}_x(\beta)$ and when the phase angle phi is π/2 the rotation is

$$\boldsymbol{R}_{xy}(90.0, \beta) = \boldsymbol{R}_{y}(\beta) \ .$$

1. Rxy(spin_sys &sys, double beta) - Returns the rotation matrix which rotates the entire spin system by the angle beta about the axis in the xy-plane phi degrees from the x axis.
2. Rxy(spin_sys &sys, int spin, double beta) - Returns the rotation matrix which rotates the spin specified by the angle beta about the axis in the xy-plane phi degrees from the x axis.
3. Rz(spin_sys &sys, char *iso, double beta) - Returns the rotation matrix which rotates the spins of the isotope type specified by the angle beta about the axis in the xy-plane phi degrees from the x axis.

The result is a spin operator as a matrix in the product (default) basis which spans the full Hilbert space of the spin system.

### Return Value:

### Example(s):

### Mathematical Basis:

For a general phase angle, $\phi$, $\boldsymbol{R}_{xy}(\phi, \beta)$ is defined mathematically to be

$$\boldsymbol{R}_{xy}(\phi, \beta) = [\boldsymbol{R}_{z}(\phi)]\boldsymbol{R}_{x}(\beta)[\boldsymbol{R}_{z}(\phi)]^{-1} \quad , \tag{18-93}$$

The explicit formula for a rotation in the xy-plane of a single spin 1/2 particle may be obtained by placement of equations (18-62) and (18-89) into equation (18-93). This produces the formula

$$\boldsymbol{R}_{xy}(\phi, \beta) = \begin{bmatrix} \cos\beta/2 & (-i\sin\beta/2)exp(-i\phi) \\ (-i\sin\beta/2)exp(i\phi) & \cos\beta/2 \end{bmatrix} \ . \tag{18-94}$$

Alternatively, the same formula can be obtained directly from equation (18-126) when the x, y, and z components of the unit vector along the rotation axis are cos$\phi$, sin$\phi$, and 0, respectively. Evaluation of these matrix elements are actually performed on

$$\boldsymbol{R}_{xy}(\phi, \beta) = \begin{bmatrix} \cos(\beta/2) & \sin(\beta/2)[-\sin\phi - i\cos\phi] \\ \sin(\beta/2)[\sin\phi - i\cos\phi] & \cos(\beta/2) \end{bmatrix} \ . \tag{18-95}$$

For spins with I>1/2, evaluation of $\boldsymbol{R}_{xy}(\phi, \beta)$ is performed directly from equation (18-93) talking the rotations about the individual axes, $\boldsymbol{R}_{z}(\phi)$ and $\boldsymbol{R}_{x}(\beta)$, from equations(18-61) and (18-58).

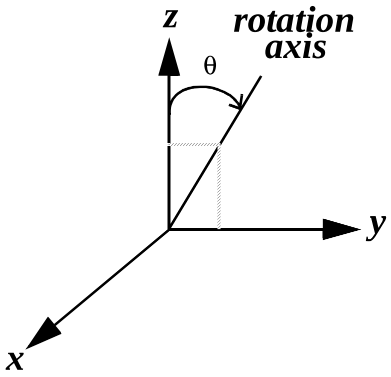
### 4.6.5 Ryz

**Usage:**

```
#include <HSLib/SpinOp.h>
spin_op Ryz (spin_sys &sys, double theta, double beta);
spin_op Ryz (spin_sys &sys, int spin, double theta, double beta);
spin_op Ryz (spin_sys &sys, char *iso, double theta, double beta);
spin_op Ryz_sp (spin_sys &sys, double theta, double beta);
```

**Description:**

The function *Ryz* returns the spin rotation operator $\mathbf{R}_{xy}(\theta,\beta)$ which rotates spin angular momentum by an angle β about an axis in the yz-plane which is θ degrees over from the z-axis. The angle β is the rotation angle, and the angle θ the phase angle as shown in the figure below.



When the phase angle θ is zero, $\boldsymbol{R}_{yz}(0, \beta) = \boldsymbol{R}_z(\beta)$ and when the phase angle theta is π/2 the rotation is $\boldsymbol{R}_{yz}(90.0, \beta) = \boldsymbol{R}_y(\beta)$ .

1. Ryz(spin_sys &sys, double beta) - Returns the rotation matrix which rotates the entire spin system by the angle beta about the axis in the yz-plane phi degrees from the z axis.
2. Ryz(spin_sys &sys, int spin, double beta) - Returns the rotation matrix which rotates the spin specified by the angle beta about the axis in the yz-plane phi degrees from the z axis.
3. Ryz(spin_sys &sys, char *iso, double beta) - Returns the rotation matrix which rotates the spins of the isotope type specified by the angle beta about the axis in the yz-plane phi degrees from the z axis.

The result is a spin operator as a matrix in the product (default) basis which spans the full Hilbert space of the spin system.

**Return Value:**

**Example(s):**

**Mathematical Basis:**

For a general phase angle, $\theta$, $\boldsymbol{R}_{yz}(\theta, \beta)$ is defined mathematically to be[1]

$$\boldsymbol{R}_{yz}(\theta, \beta) = [\boldsymbol{R}_x(\theta)]^{-1}\boldsymbol{R}_z(\beta)[\boldsymbol{R}_x(\theta)] \quad , \tag{18-96}$$

The explicit formula for a rotation in the yz-plane of a single spin 1/2 particle may be obtained by placement of equations (18-76) and (18-89) into equation (18-93). Alternatively, the same formula can be obtained directly from equation (18-126) when the x, y, and z components of the unit vector along the rotation axis are 0, sinθ, and cosθ, respectively. Either procedure produces the formula

$$\boldsymbol{R}_{yz}(\theta, \beta) = \begin{bmatrix} \cos(\beta/2) - i\sin(\beta/2)\cos\theta & -\sin(\beta/2)\sin\theta \\ \sin(\beta/2)\sin\theta & \cos(\beta/2) + i\sin(\beta/2)\cos\theta \end{bmatrix} \quad . \tag{18-97}$$

For spins with I>1/2, evaluation of $\boldsymbol{R}_{yz}(\phi, \beta)$ is performed directly from equation (18-93) talking the rotations about the individual axes, $\boldsymbol{R}_z(\beta)$ and $\boldsymbol{R}_y(\theta)$ , from equations(18-61) and (18-76).

## 4.6.6 Rzx

### Usage:

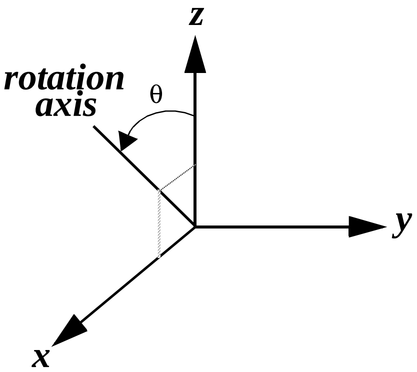
```
#include <HSLib/SpinOp.h>
spin_op Rzx (spin_sys &sys, double theta, double beta);
spin_op Rzx (spin_sys &sys, int spin, double theta, double beta);
spin_op Rzx (spin_sys &sys, char *iso, double theta, double beta);
spin_op Rzx_sp (spin_sys &sys, double theta, double beta);
```

### Description:

The function *Rzx* returns the spin rotation operator $\mathbf{R}_{zx}(\theta,\beta)$ which rotates spin angular momentum by an angle β about an axis in the zx-plane which is θ degrees down from the z-axis. The angle β is the rotation angle,

[1] Note the in this equation the $\boldsymbol{R}_x(\theta)$ operators "sandwich" $\boldsymbol{R}_z(\beta)$ in the reverse order than the "sandwich" of $\boldsymbol{R}_x(\beta)$ by $\boldsymbol{R}_z(\phi)$ in equation (18-93). This is solely because we have used the angle $\theta$ to coincide with the polar angle in a right handed spherical coordinate system.

and the angle θ the phase angle as shown in the figure below.



When the phase angle ϕ is zero, $\boldsymbol{R}_{zx}(0, \beta) = \boldsymbol{R}_z(\beta)$ and when the phase angle theta is π/2 the rotation is $\boldsymbol{R}_{zx}(90.0, \beta) = \boldsymbol{R}_x(\beta)$ .

1. Rzx(spin_sys &sys, double beta) - Returns the rotation matrix which rotates the entire spin system by the angle beta about the axis in the zx-plane theta degrees from the z axis.
2. Rzx(spin_sys &sys, int spin, double beta) - Returns the rotation matrix which rotates the spin specified by the angle beta about the axis in the zx-plane theta degrees from the z axis.
3. Rzx(spin_sys &sys, char *iso, double beta) - Returns the rotation matrix which rotates the spins of the isotope type specified by the angle beta about the axis in the zx-plane theta degrees from the z axis.

The result is a spin operator as a matrix in the product (default) basis which spans the full Hilbert space of the spin system.

## Return Value:

## Example(s):

## Mathematical Basis:

For a general phase angle, $\theta$, $\boldsymbol{R}_{zx}(\theta, \beta)$ is defined mathematically to be

$$\boldsymbol{R}_{zx}(\theta, \beta) = [\boldsymbol{R}_y(\theta)]\boldsymbol{R}_z(\beta)[\boldsymbol{R}_y(\theta)]^{-1} \quad , \tag{18-98}$$

The explicit formula for a rotation in the zx-plane of a single spin 1/2 particle may be obtained by placement of equations ??? and ??? into equation (18-93). Alternatively, the same formula can be obtained directly from equation (18-126) when the x, y, and z components of the unit vector along the rotation axis are sinθ, 0, and cosθ, respectively.This produces the formula

$$\boldsymbol{R}_{zx}(\theta, \beta) = \begin{bmatrix} \cos\beta/2 - i\cos\theta\sin\beta/2 & -i\sin\theta\sin\beta/2 \\ -i\sin\theta\sin\beta/2 & \cos\beta/2 + i\cos\theta\sin\beta/2 \end{bmatrix}. \tag{18-99}$$

For spins with I>1/2, evaluation of $\boldsymbol{R}_{zx}(\theta, \beta)$ is performed directly from equation (18-93) talking the rotations about the individual axes, $\boldsymbol{R}_z(\phi)$ and $\boldsymbol{R}_y(\beta)$ , from equations(18-61) and ???.

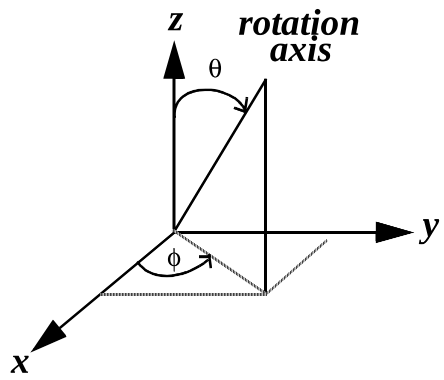
### 4.6.7 Rxyz

**Usage:**

```
#include <HSLib/SpinOp.h>
spin_op Rxyz(spin_sys&, double theta, double phi, double beta);
spin_op Rxyz(spin_sys&, char*, double theta, double phi, double beta);
spin_op Rxyz(spin_sys&, int, double theta, double phi, double beta);
spin_op Rxyz_sp (spin_sys &sys, double theta, double phi, double beta);
```

**Description:**

The function *Rxyz* returns the spin rotation operator $\mathbf{R}_{xyz}(\theta,\phi,\beta)$ which rotates spin angular momentum by an angle $\beta$ about an axis which is $\theta$ degrees down from the z-axis and $\phi$ degrees over from the x-axis. The angle $\beta$ is the rotation angle, and the angle $\theta$ and $\phi$ the polar angle as shown in the figure below.



When the angles $\theta$ and $\phi$ are zero, $\boldsymbol{R}_{xyz}(0, 0, \beta) = \boldsymbol{R}_z(\beta)$ . Other equalities which relate to previously defined functions are $\boldsymbol{R}_{xyz}\left(\frac{\pi}{2}, 0, \beta\right) = \boldsymbol{R}_x(\beta)$ and $\boldsymbol{R}_{xyz}\left(0, \frac{\pi}{2}, \beta\right) = \boldsymbol{R}_y(\beta)$ for rotations about the coordinate axes and $\boldsymbol{R}_{xyz}\left(\frac{\pi}{2}, \phi, \beta\right) = \boldsymbol{R}_{xy}(\beta)$ , $\boldsymbol{R}_{xyz}\left(\theta, \frac{\pi}{2}, \beta\right) = \boldsymbol{R}_{yz}(\beta)$ and $\boldsymbol{R}_{xyz}(\theta, 0, \beta) = \boldsymbol{R}_{zx}(\beta)$ for the rotations in the three coordinate planes.

1. Rxyz(spin_sys &sys, double theta, double phi, double beta) - Returns the rotation matrix which rotates the entire spin system by the angle beta about the axis theta degrees down from the z axis and phi degrees over from the x axis.
2. Rxyz(spin_sys &sys, int spin, double theta, double phi, double beta) - Returns the rotation matrix which rotates the spin specified by the angle beta about the axis theta degrees down from the z axis and phi degrees over from the x axis.
3. Rxyz(spin_sys &sys, char *iso, double theta, double phi, double beta) - Returns the rotation matrix which rotates the spins of the isotope type specified by the angle beta about the axis theta degrees down from

the z axis and phi degrees over from the x axis.

The result is a spin operator as a matrix in the product (default) basis which spans the full Hilbert space of the spin system.

### Return Value:

### Example(s):

### Mathematical Basis:

The general formula for a rotation about an arbitrary axis for a single spin 1/2 particle is derived elsewhere. A more explicit version can be obtained directly from equation (18-126) when the x, y, and z components of the unit vector along the rotation axis are (see the previous figure) sinθcosφ, sinθsinφ, and cosθ, respectively.This produces the formula

$$\boldsymbol{R}_{xyz}(\theta, \phi, \beta) = \begin{bmatrix} \cos\frac{\beta}{2} - i\cos\theta\sin\frac{\beta}{2} & (-i\sin\theta\cos\phi - \sin\theta\sin\phi)\sin\frac{\beta}{2} \\ (-i\sin\theta\cos\phi + \sin\theta\sin\phi)\sin\frac{\beta}{2} & \cos\frac{\beta}{2} + i\cos\theta\sin\frac{\beta}{2} \end{bmatrix}, \quad (18\text{-}100)$$

or equivalently,

$$\boldsymbol{R}_{xyz}(\theta, \phi, \beta) = \begin{bmatrix} \cos\frac{\beta}{2} - i\cos\theta\sin\frac{\beta}{2} & -iexp(-i\varphi)\sin\theta\sin\frac{\beta}{2} \\ -iexp(i\phi)\sin\theta\sin\frac{\beta}{2} & \cos\frac{\beta}{2} + i\cos\theta\sin\frac{\beta}{2} \end{bmatrix}. \quad (18\text{-}101)$$

From geometric arguments we can specify this rotation in terms of three individual rotations, one for each of the angles used in the function call.

$$\boldsymbol{R}_{xyz}(\theta, \phi, \beta) = [\boldsymbol{R}_z(\phi)]^{-1}[\boldsymbol{R}_y(\theta)]^{-1}\boldsymbol{R}_z(\beta)\boldsymbol{R}_y(\theta)\boldsymbol{R}_z(\phi) \quad (18\text{-}102)$$

For spins with I>1/2, evaluation of $\boldsymbol{R}_{xyz}(\theta, \phi, \beta)$ is performed directly from equation (18-102) talking the rotations about the individual axes, $\boldsymbol{R}_z(\phi)$ and $\boldsymbol{R}_y(\beta)$ , from equations(18-61) and ???.

## 4.6.8 R_Euler

### Usage:

```
#include <HSLib/SpinOp.h>
spin_op R_Euler(spin_sys&, double alpha, double beta, double gamma);
spin_op R_Euler(spin_sys&, int spin, double alpha, double beta, double gamma);
spin_op R_Euler(spin_sys&, char *iso, double alpha, double beta, double gamma);
spin_op R_Euler_sp(spin_sys&, double alpha, double beta, double gamma);
```

### Description:

The first three instances of the function rotate return rotation matrices with given phase and flip angle about an axis in the x-y plane. The first call returns a matrix rotating all the spins. The second call takes a string as the second argument specifying the type of the nuclei that have to be affected by this matrix. This is to be used for simulating spin systems with heteronuclei.

The second three functions return rotations parameterized by the three Euler angles. The returned matrices rotate a given spin, spins of a given sort or the whole spin system.

1. rotate(spin_sys &sys, double phi, double beta) - Returns the rotation matrix which rotates the spin system by the angle beta about the axis which is the angle phi over from the x-axis in the xy-plane.
2. rotate(spin_sys &sys, char *iso, double phi, double beta) - Returns the rotation matrix which rotates the spins in the system of isotope type given by the angle beta about the axis which is the angle phi over from the x-axis in the xy-plane
3. rotate(spin_sys &sys, char *iso, double phi, double beta) - Returns the rotation matrix which rotates the spins in the system of isotope type given by the angle beta about the axis which is the angle phi over from the x-axis in the xy-plane

### Return Value:

The rotation matrix in the product spin function base.

### Example(s):

```
#include <gamma.h>
main()
  {
  spin_sys ab(2);
  spin_op op1(ab), op2(ab), op3(ab);
  op1=rotate(ab,PI/2,PI/2);              // non-selective 90-degree y pulse
  op2=rotate(ab,”1H”,0,PI);              // 180-x pulse on the protons
  op3=rotate(ab,0,0,PI);                 // 180-x pulse on the first spin
  op1=rotate(ab,alpha,beta,gamma);       // rotate with Euler angles
  }
```

### See Also:

# 4.7 Rotated Spin Operators

## 4.7.1 Ixy

**Usage:**

```
#include <HSLib/SpinOp.h>
spin_op Ixy(spin_sys &sys, int spin, double phi);
```

**Description:**

The function *Ixy* returns a spin operator which is the operator $I_{ix}$ rotated about the z-axis by angle phi

**Return Value: A spin operator**

**Example(s):**

**Mathematical Basis:**

## 4.7.2 Fxy

**Usage:**

```
#include <HSLib/SpinOp.h>
spin_op Fxy(spin_sys &sys, double phi);
spin_op Fxy(spin_sys &sys, int spin, double phi);
spin_op Fxy(spin_sys &sys, char *iso, double phi);
spin_op Fxy_sp(spin_sys &sys, double phi);
```

**Description:**

The function *Ixy* returns a spin operator which is the operator $I_{ix}$ rotated about the z-axis by angle phi

**Return Value: A spin operator**

**Example(s):**

**Mathematical Basis:**

### 4.7.3 Fp

**Usage:**

```
#include <HSLib/SpinOp.h>
spin_op Fp(spin_sys &sys, double phi);
spin_op Fp(spin_sys &sys, int spin, double phi);
spin_op Fp(spin_sys &sys, char *iso, double phi);
spin_op Fp_sp(spin_sys &sys, double phi);
```

**Description:**

The function *Fp* returns a spin operator which is the operator $I_{ix}$ rotated about the z-axis by angle phi

**Return Value: A spin operator**

**Example(s):**

**Mathematical Basis:**

### 4.7.4 Fm

**Usage:**

```
#include <HSLib/SpinOp.h>
spin_op Fm(spin_sys &sys, double phi);
spin_op Fm(spin_sys &sys, int spin, double phi);
spin_op Fm(spin_sys &sys, char *iso, double phi);
spin_op Fm_sp(spin_sys &sys, double phi);
```

**Description:**

The function *Fm* returns a spin operator which is the operator $I_{ix}$ rotated about the z-axis by angle phi

**Return Value: A spin operator**

**Example(s):**

**Mathematical Basis:**

# 4.8 Spin Operator General Functions

## 4.8.1 adjoint

**Usage:**

```
#include <HSLib/SpinOp.h>
spin_op adjoint(spin_op &SOp);
```

**Description:**

The function *adjoint* returns a spin operator which is the adjoint of the spin operator given as an argument.

**Return Value: A spin operator which is the adjoint of the spin operator given.**

**Example(s):**

**Mathematical Basis:**

## 4.8.2 trace

**Usage:**

```
#include <HSLib/SpinOp.h>
complex trace(spin_op &SOp);
```

**Description:**

The function *trace* returns the trace of the spin operator matrix given in the argument.

**Return Value: A complex number which is the trace of the spin operator given.**

**Example(s):**

**Mathematical Basis:**

The trace of a spin operator is given by

$$Tr\{\boldsymbol{SOp}\} = \sum_{i}^{dim} \langle i|\boldsymbol{SOp}|i\rangle \tag{18-103}$$

where dim is the dimension of the composite Hilbert space of the spin system associated with SOp. Although slightly more expensive computationally, the trace of a spin operator can also be performed on the spin operator expressed in the single spin Hilbert space form *via* the formula[1]

$$Tr\{\boldsymbol{SOp}\} = Tr\{\boldsymbol{SOp}(1) \otimes \boldsymbol{SOp}(2) \otimes \ldots \otimes \boldsymbol{SOp}(n)\} = \prod_{n}^{spins} Tr\{\boldsymbol{SOp}(n)\} \tag{18-104}$$

where $\boldsymbol{SOp}(n)$ is spin n's contribution to the operator $\boldsymbol{SOp}$ in its own Hilbert space and the symbol $\otimes$ is used to indicate a direct product. This function will preferentially use (18-103). Equation (18-104) will be

[1] See Blum, K., page 201, equation A2b.

used when $\boldsymbol{SOp}$ exists only in the compact single spin space form, thus avoiding expanding the spin operator into the spin system composite space.

## 4.9 Description

Class ***SpinOp (spin_op)*** contains all the properties of operators specifically associated with the spins in a spin system. *Spin operators* has an intrinsic knowledge of common spin operators such as $\mathbf{I}_x$, $\mathbf{I}_y$, $\mathbf{I}_z$, $\mathbf{I}_+$, $\mathbf{I}_-$, etc. This class also knows about operators which rotate spin angular momentum and spherical tensor operators of differing ranks. Internally, *Class SPIN OPERATOR* knows how to take direct products of single spin operators to produce spin operators which act on multiple spins. This last feature allows for efficient use of memory and faster spin operator computations.

**Full Hilbert Space Matrix**: One way to represent a spin operator, **SOp**, is that of a matrix (*see Class MATRIX*) in the product basis (used as the default basis). The figure below shows a typical matrix form of a spin operator,

**SOp Matrix (SOp)** **SOp Basis (U)**

**Product Basis Representation**

$$\begin{bmatrix} 11 & 12 & 13 & \cdots & 1N \\ 21 & 22 & 23 & \cdots & 2N \\ 31 & 32 & 33 & \cdots & 3N \\ \cdot & \cdot & \cdot & & \cdot \\ \cdot & \cdot & \cdot & & \cdot \\ \cdot & \cdot & \cdot & & \cdot \\ \cdot & \cdot & \cdot & & \cdot \\ N1 & N2 & N3 & \cdots & NN \end{bmatrix} \qquad \begin{bmatrix} 1 & & & & & & \\ & 1 & & & & \mathbf{0} & \\ & & 1 & & & & \\ & & & \ddots & & & \\ & & & & 1 & & \\ & \mathbf{0} & & & & 1 & \\ & & & & & & 1 \end{bmatrix}$$

where N is the Hilbert space dimension of the spin system, $N = \prod_{i}^{spins}[2\boldsymbol{I}_i + 1]$ with $\mathbf{I}_i$ the spin quantum number of spin i. This array, SOp, is what GAMMA will output when asked to print an operator. The relationship between a spin operator, **SOp**, in the default basis (superscript DB) and the same operator in an arbitrary basis (superscript AB) is given by the following equation.

$$(\boldsymbol{U}^{\boldsymbol{AB}})^{\dagger}\boldsymbol{SOp}^{\boldsymbol{AB}}\boldsymbol{U}^{\boldsymbol{AB}} = \boldsymbol{SOp}^{\boldsymbol{DB}} \qquad \textbf{(18-105)}$$

Here **U** is the transformation matrix or basis matrix (see *Class BASIS*) and **SOp** the spin operator matrix. Spin operators are invariably stored in the default basis (**U** = **1** is not stored).

**Single Spin Matrices**: An alternative way to represent a spin operator is in terms of single spin operators. Each single spin operator in this context is stored in the spin Hilbert space of dimension $[2\boldsymbol{I}_i + 1]$, not in the full spin system Hilbert space. The full Hilbert space representation is obtained from these single spin matrices by taking direct products.

$$\boldsymbol{SOp} = \boldsymbol{SOp}(1) \otimes \boldsymbol{SOp}(2) \otimes \boldsymbol{SOp}(3) \otimes \ldots \otimes \boldsymbol{SOp}(n) \qquad \textbf{(18-106)}$$

In this equation, $\boldsymbol{SOp}(i)$ is the single spin operator acting on spin i. All spins which are active in $\boldsymbol{SOp}$ are included in the product and each single spin operator here spans its own Hilbert Space.

Although not generally possible for any arbitrary spin operators[1], use of this operator description can result in both computational and memory savings. The memory savings can be made evident by consideration of the spin operator $\mathbf{I}_{1x}$ in a three spin system with spins having quantum numbers of 1/2, 1, and 1/2 respectively. The full Hilbert space of this spin system is 12 resulting from the product of the individual spin sub-spaces of 2, 3, and 2. The operator then looks like

$$\boldsymbol{I}_{1x} = \begin{bmatrix} 12 \times 12 \\ Array \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \otimes \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \otimes \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}. \qquad (18\text{-}107)$$

or equivalently

$$\boldsymbol{I}_{1x} = \boldsymbol{I}_{1x} \otimes \boldsymbol{I}_{2e} \otimes \boldsymbol{I}_{3e} \qquad \mathbf{(18\text{-}108)}$$

For this simple three spin system, the full Hilbert space representation is a 12 x 12 array whereas by the individual spin matrices it is significantly smaller. GAMMA wisely stores only the diagonal elements in a diagonal array, thus internally storing only 9 elements rather than 144 for this array. Computations occur along a similar vein, when GAMMA can work with this sub-space representations it will do so preferentially, hence running significantly faster than when working with the expanded form.

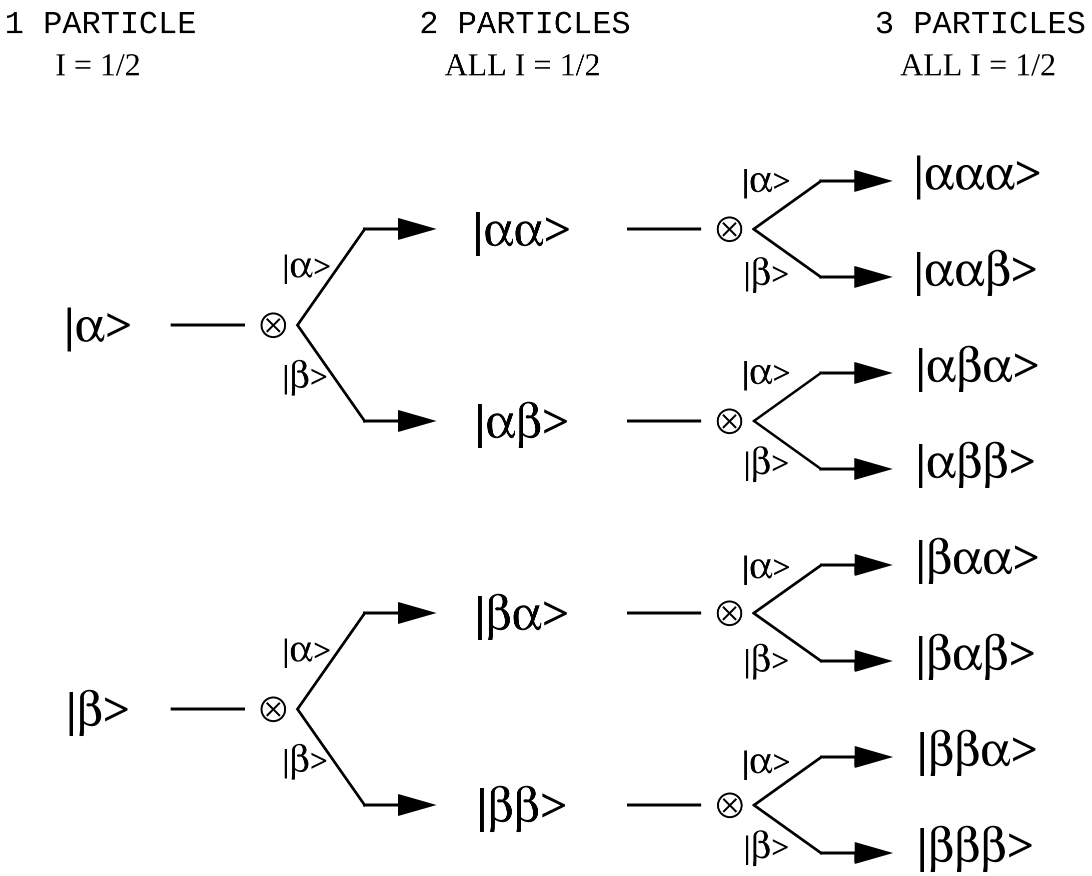
*Basis Ordering:* As previously mentioned, spin operators are invariably maintained in the default basis, the product basis, and of course this in part dictates how the matrix representations of SOp’s will look. Spin Operator adheres to three rules in setting up the product basis -

1 Spins are ordered in accordance with the defined spin system of which they are a part.

2 Spin states are ordered from high to low $I_z$ values.

3 The spins indexed latest change the most rapidly.

[1] The question immediately arises as to when a spin operator (or any operator) can be expressed in terms of direct products. This will be true when the operator can be broken up into products of mutually commuting operators, such as the set $\{\mathbf{I}_{ix}\}$ for a spin system which all must commute as they act on different spins.



It is then clear how these functions are labeled. The Iz operators for a two spin 1/2 system have matrix representations in the composite Hilbert space of the spin system as follows

$$I_{1z} = \frac{1}{2}\begin{vmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & -1 \end{vmatrix} \quad \text{and that} \quad I_{2z} = \frac{1}{2}\begin{vmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{vmatrix} \qquad (18\text{-}109)$$

The transformation between the different representations of a spin operator is handled entirely within *Class OPERATOR*. Most calculations can make use of spin operators without knowledge of how GAMMA is managing operator representations.

## 4.9.1 Single Spin Ix Operator

## 4.9.2 Spin Rotation Operators:

*General* - A general rotation of angular momentum **J** of angle β about an axis **u** is defined as

$$\boldsymbol{R}_u(\beta) = exp[-i\beta(\boldsymbol{u}\bullet\boldsymbol{J})] = exp(-i\beta\boldsymbol{J}_u) \qquad (18\text{-}110)$$

where $\boldsymbol{J}_u$ is the component of angular momentum along the axis u, and $\boldsymbol{u}$ a unit vector along the axis u.

*Single Spin I=1/2* - For a single spin, i, the rotation of spin angular momentum is then

$$\boldsymbol{R}_{iu}(\beta) = exp(-i\beta\boldsymbol{I}_{iu}) . \qquad (18\text{-}111)$$

Explicit formulae for rotations of spin 1/2 species about any arbitrary axis is obtainable through use of the relationship between the single spin operators and the Pauli spin matrices below[1].

$$\sigma_x = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \qquad \sigma_y = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix} \qquad \sigma_z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \qquad (18\text{-}112)$$

Direct comparison with equations (18-8), (18-14), and (18-22) to the previous equation demonstrates that[2] for I = 1/2

$$\boldsymbol{I}_{iu}^{(1/2)} = \frac{1}{2}\sigma_u \text{ where } u \in \{x, y, z\} . \qquad (18\text{-}113)$$

In fact, equation (18-113) is valid for any axis $u$, not just the coordinate axes x, y and z. This is shown by expanding the general axis case in terms of the x, y, and z components.

$$\boldsymbol{I}_{iu}^{(1/2)} = u_x\boldsymbol{I}_{ix}^{(1/2)} + u_y\boldsymbol{I}_{iy}^{(1/2)} + u_z\boldsymbol{I}_{iz}^{(1/2)} \qquad (18\text{-}114)$$

$$\boldsymbol{I}_{iu}^{(1/2)} = \frac{1}{2}[u_x\sigma_x + u_y\sigma_y + u_z\sigma_z] = \frac{1}{2}\sigma_u \qquad (18\text{-}115)$$

In these equations, $u_x$, $u_y$, and $u_z$ are the components of the *unit vector* in the direction of axis $u$. An alternative form of (18-115) is

$$\boldsymbol{I}_{iu}^{(1/2)} = \frac{1}{2}(\boldsymbol{u}\bullet\sigma) \qquad (18\text{-}116)$$

It is easily verified from (18-115) and (18-122) that the Pauli matrices are idempotent, that is, they have the property

$$\sigma_u^2 = 1 . \qquad (18\text{-}117)$$

In turn, the single spin (I = 1/2) operators have the properties

[1] See Blum, pg. 4, Eq. (1.1.6), Schiff pg. 205, Eq. (27.30), and/or Gottfried, pg 275, Eq. (12).

[2] Note that the matrix forms of $\boldsymbol{I}_{ix}$, $\boldsymbol{I}_{iy}$, and $\boldsymbol{I}_{iz}$ used for comparison are in the single spin Hilbert space and independent of the index i.

$$[\boldsymbol{I}_{iu}^{(1/2)}]^{2m} = [1/2]^{2m}\mathbf{1} \tag{18-118}$$

and

$$[\boldsymbol{I}_{iu}^{(1/2)}]^{2m+1} = (1/2)^{2m}\boldsymbol{I}_{iu}^{(1/2)} = 2[1/2]^{2m+1}\boldsymbol{I}_{iu}^{(1/2)} \quad , \tag{18-119}$$

where m is an integer[1]. We now turn our attention back to the derivation of formulae for the rotations of spin 1/2 species about any arbitrary axis. From equation (18-111), we desire

$$\boldsymbol{R}_{iu}^{(1/2)}(\beta) = exp[-i\beta\boldsymbol{I}_{iu}^{(1/2)}] = \cos[\beta\boldsymbol{I}_{iu}^{(1/2)}] - i\sin[\beta\boldsymbol{I}_{iu}^{(1/2)}] \quad . \tag{18-120}$$

Applying the series expansions of sine and cosine,

$$\boldsymbol{R}_{iu}^{(1/2)}(\beta) = \sum_m (-1)^m \frac{[\beta\boldsymbol{I}_{iu}^{(1/2)}]^{2m}}{(2m)!} - i\sum_m (-1)^m \frac{[\beta\boldsymbol{I}_{iu}^{(1/2)}]^{2m+1}}{(2m+1)!} \quad , \tag{18-121}$$

followed by the use of equations (18-118) and (18-119),

$$\boldsymbol{R}_{iu}^{(1/2)}(\beta) = \mathbf{1}\sum_m (-1)^m \frac{(\beta/2)^{2m}}{(2m)!} - i2\boldsymbol{I}_{iu}^{(1/2)}\sum_m (-1)^m \frac{(\beta/2)^{2m+1}}{(2m+1)!} \quad , \tag{18-122}$$

produces a more compact formula,

$$\boldsymbol{R}_{iu}^{(1/2)}(\beta) = \mathbf{1}\cos(\beta/2) - i2\boldsymbol{I}_{iu}^{(1/2)}\sin(\beta/2) \quad . \tag{18-123}$$

Using equation (18-115), this becomes

$$\boldsymbol{R}_{iu}^{(1/2)}(\beta) = \mathbf{1}\cos(\beta/2) - i\sigma_u\sin(\beta/2) \quad . \tag{18-124}$$

Explicitly, we have

$$\boldsymbol{R}_{iu}^{(1/2)}(\beta) = \cos(\beta/2)\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} - i\sin(\beta/2)\left\{u_x\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} + u_y\begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix} + u_z\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}\right\} \tag{18-125}$$

The general formula for a spin 1/2 particle is then[2]

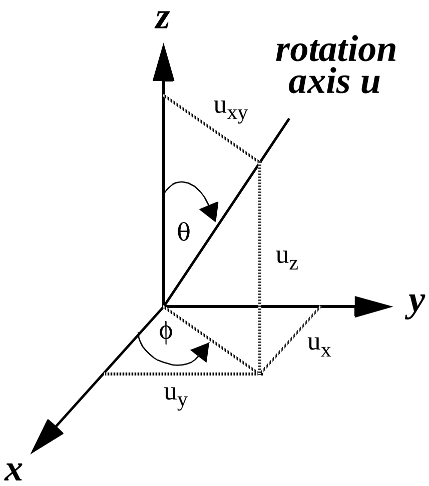
$$\boldsymbol{R}_{iu}^{(1/2)}(\beta) = \begin{bmatrix} \cos\frac{\beta}{2} - iu_z\sin\frac{\beta}{2} & (-iu_x - u_y)\sin\frac{\beta}{2} \\ (-iu_x + u_y)\sin\frac{\beta}{2} & \cos\frac{\beta}{2} + iu_z\sin\frac{\beta}{2} \end{bmatrix} \quad . \tag{18-126}$$

where $u_x$, $u_y$, and $u_z$ are the components of the unit vector pointing in the direction of **u** and $\beta$ is

[1] Note that there is a mismatch of “powers to the m” between the left and right hand sides of equation (18-119). This must be so, as seen from the m=0 case, and results in the factor of 2 on the sine term of (18-123).

[2] See Cohen-Tannoudji, Volume Two, pages 983-985, Sakurai, pages 165-166, or Goldman page 64.

the angle of rotation. The following diagram shows these components with respect to the standard spherical coordinate system.



$$u_z = \cos\theta,\ u_{xy} = \sin\theta,\ u_x = u_{xy}\cos\phi = \sin\theta\cos\phi, \text{ and } u_y = u_{xy}\sin\phi = \sin\theta\sin\phi \quad (18\text{-}127)$$

Thus equation (18-126) can be recast as

$$\boldsymbol{R}_{iu}^{(1/2)}(\beta) = \begin{bmatrix} \cos\frac{\beta}{2} - i\cos\theta\sin\frac{\beta}{2} & (-i\cos\phi - \sin\phi)\sin\theta\sin\frac{\beta}{2} \\ (-i\cos\phi + \sin\phi)\sin\theta\sin\frac{\beta}{2} & \cos\frac{\beta}{2} + i\cos\theta\sin\frac{\beta}{2} \end{bmatrix} \quad (18\text{-}128)$$

or

$$\boldsymbol{R}_{iu}^{(1/2)}(\beta) = \begin{bmatrix} \cos\frac{\beta}{2} - i\cos\theta\sin\frac{\beta}{2} & -ie^{-i\phi}\sin\theta\sin\frac{\beta}{2} \\ -ie^{i\phi}\sin\theta\sin\frac{\beta}{2} & \cos\frac{\beta}{2} + i\cos\theta\sin\frac{\beta}{2} \end{bmatrix}. \quad (18\text{-}129)$$

*Single Spin I>1/2* - The rotation formulas involving $\boldsymbol{R}_{iu}^{(1/2)}$ are exclusively applicable to I=1/2 spins because equation (18-113) is not valid for I>1/2. In cases involving spins with I > 1/2 one is forced to use the general formula, equation (18-111), which is reproduced below.

$$\boldsymbol{R}_{iu}(\beta) = exp(-i\beta\boldsymbol{I}_{iu})\,. \quad (18\text{-}130)$$

Not only is this more difficult to directly evaluate, it is computationally more expensive to perform. Fortunately, one may usually work with $\boldsymbol{R}_{in}(\beta)$ in the Hilbert space of the spin itself and, for example in the case of a spin with **I**=1 will only be a (3x3) matrix. Expanding $\boldsymbol{I}_{iu}$ in the previous formula with the equation analogous to (18-114) without regard to the spin quantum number,

$$\boldsymbol{I}_{iu} = u_x\boldsymbol{I}_{ix} + u_y\boldsymbol{I}_{iy} + u_z\boldsymbol{I}_{iz} \tag{18-131}$$

produces

$$\boldsymbol{R}_{iu}(\beta) = exp(-i\beta\boldsymbol{I}_{iu}) = exp[-i\beta(u_x\boldsymbol{I}_{ix} + u_y\boldsymbol{I}_{iy} + u_z\boldsymbol{I}_{iz})] \tag{18-132}$$

From the previous trigonometric relationships of (18-127) this is equivalent to

$$\boldsymbol{R}_{iu}(\beta) = exp\{-i\beta[(\sin\theta\cos\phi)\boldsymbol{I}_{ix} + (\sin\theta\sin\phi)\boldsymbol{I}_{iy} + \cos\theta\boldsymbol{I}_{iz}]\} \tag{18-133}$$

$$\boldsymbol{R}_{iu}(\beta) = e^{(-i\beta\sin\theta\cos\phi\boldsymbol{I}_{ix})}e^{(-i\beta\sin\theta\sin\phi\boldsymbol{I}_{iy})}e^{(-i\beta\cos\theta\boldsymbol{I}_{iz})} \tag{18-134}$$

Still under construction ********* SOSI (18-135)

$$\boldsymbol{R}_{iu}(\beta) = \boldsymbol{R}_{iz}(\phi)\boldsymbol{R}_{iy}(-\theta)\boldsymbol{R}_{iz}(\beta)[\boldsymbol{R}_{iy}(-\theta)]^{-1}[\boldsymbol{R}_{iz}(\phi)]^{-1} \tag{18-136}$$

*Multi-spin Rotations* - We now look into rotations performed on systems containing multiple spins. This would perhaps be the rotation of the total spin angular momentum in a multispin system where one desires

$$\boldsymbol{R}_{u}(\beta) = exp(-i\beta\boldsymbol{F}_{u}) \tag{18-137}$$

with $\boldsymbol{F}_u$ the component of the total spin angular momentum operator along the axis **u**[1]. One might also desire a specific set of spins in a multispin system to be rotated. The operator form will be identical, only the spin angular momentum operator will change.

$$\boldsymbol{R}_{\{i\}u}(\beta) = exp(-i\beta\boldsymbol{F}_{\{i\}u}) \tag{18-138}$$

In this case {i} indicates the set of i spins that one rotates[2]. Equation (18-138) is equivalent to equation (18-137) when all spins in the system are included in {i}, it is equivalent to (18-111) when only one spin is included in {i}. Expansion of (18-138) produces

$$\boldsymbol{R}_{\{i\}u}(\beta) = exp(-i\beta\boldsymbol{F}_{\{i\}u}) = exp\left|\sum_{\{i\}}^{spins} -i\beta\boldsymbol{I}_{iu}\right| \tag{18-139}$$

and since all of the single spin operators commute,

$$\boldsymbol{R}_{\{i\}u}(\beta) = \prod_{\{i\}}^{spins} exp(-i\beta\boldsymbol{I}_{iu}) = \prod_{\{i\}}^{spins}\boldsymbol{R}_{iu}(\beta) \quad . \tag{18-140}$$

[1] The mathematical formula that applies here is $\boldsymbol{F}_u = \boldsymbol{u}\bullet\boldsymbol{F} = \sum_i^{spins}[\boldsymbol{u}\bullet\boldsymbol{I}_i] = \sum_i^{spins}\boldsymbol{I}_{iu}$ which is equivalent to previous definitions in the cases where $u \in \{x, y, z\}$.

[2] The formula defining $\boldsymbol{F}_{\{i\}u}$ is analogous to that in the previous footnote but with the sum only containing spins specifically in the set {i}.

Another useful feature of all single spin operators commuting is that the products in equation (18-140) can be attained with the use of direct products[1].

$$\boldsymbol{R}_{\{i\}u}(\beta) = exp(-i\beta\delta_{1\{i\}}\boldsymbol{I}_{1u}) \otimes exp(-i\beta\delta_{2\{i\}}\boldsymbol{I}_{2u}) \otimes \ldots \otimes exp(-i\beta\delta_{N\{i\}}\boldsymbol{I}_{Nu}) \tag{18-141}$$

$$\boldsymbol{R}_{\{i\}u}(\beta) = \boldsymbol{R}_{1u}(\beta\delta_{1\{i\}}) \otimes \boldsymbol{R}_{2u}(\beta\delta_{2\{i\}}) \otimes \ldots \otimes \boldsymbol{R}_{Nu}(\beta\delta_{N\{i\}}) \tag{18-142}$$

In this formula, there are terms for each spin in an N spin system and the value of $\delta_{j\{i\}}$ is zero when j is a spin not in the set {i} and 1 when it is in {i}. Although equation (18-142) may appear more foreboding than (18-140), it is much more efficient in computing $\boldsymbol{R}_{\{i\}u}(\beta)$ and use less memory to store it.

[1] See Ernst, Bodenhausen, and Wokaun, page 405, equation (8.1.5).

## 4.10 Implicit Spin Operators

### Single Spin Operators

```
spin_op Ie(spin_sys&, int);
spin_op Ix(spin_sys&, int);
spin_op Iy(spin_sys&, int);
spin_op Iz(spin_sys&, int);
spin_op Ip(spin_sys&, int);
spin_op Im(spin_sys&, int);
spin_op Ia(spin_sys&, int);
spin_op Ib(spin_sys&, int);
```

### Isotope Spin Operators

```
spin_op Fe(spin_sys&, char*);
spin_op Fx(spin_sys&, char*);
spin_op Fy(spin_sys&, char*);
spin_op Fz(spin_sys&, char*);
spin_op Fp(spin_sys&, char*);
spin_op Fm(spin_sys&, char*);
spin_op Fa(spin_sys&, char*);
spin_op Fb(spin_sys&, char*);
```

### Isotope Spin Rotation Operators

```
spin_op Rx(spin_sys&, char*, double);
spin_op Ry(spin_sys&, char*, double);
spin_op Rz(spin_sys&, char*, double);
```

### Total Spin Operators

```
spin_op Fe(spin_sys&);
spin_op Fx(spin_sys&);
spin_op Fy(spin_sys&);
spin_op Fz(spin_sys&);
spin_op Fp(spin_sys&);
spin_op Fm(spin_sys&);
spin_op Fa(spin_sys&);
spin_op Fb(spin_sys&);
```

### Single Spin Rotation Operators

```
spin_op Rx(spin_sys&, int, double);
spin_op Ry(spin_sys&, int, double);
spin_op Rz(spin_sys&, int, double);
```

### Total Spin Rotation Operators

```
spin_op Rx(spin_sys&, double);
spin_op Ry(spin_sys&, double);
spin_op Rz(spin_sys&, double);
```

### Single Spin Rotation Operators

```
spin_op Rx(spin_sys&, int, double);
spin_op Ry(spin_sys&, int, double);
spin_op Rz(spin_sys&, int, double);
```

# 5 Class Basis

## 5.1 Overview

The class ***basis*** maintains the different bases necessary for the algebraic manipulations of matrix based data types (operators, superoperators and matrices). Since operators are fundamentally associated with bases, class ***operator*** internally handles bases for each operator through use of class ***basis***. Users of GAMMA need not deal directly with class ***basis*** for any operator computations, therefore in many instances class ***basis*** will be completely transparent. However, the internal structure of class ***basis*** knows nothing about operators and any basis can be used as an independent quantity to directly manipulate matrices if the user so chooses.

Class ***basis*** contains routines to create bases, compare bases and to convert an operator (in matrix representation) from its representation in one basis into its representation in another. A single operator may be expressed in several bases and/or there may be several operators associated with the same basis. If a basis is copied to another basis, only a reference is copied, not the whole data.

Each basis consists of a transformation matrix, ***U***, which will convert an operator ***Op*** between the basis itself and the default basis, as given by the similarity transformation specified by equation

$$\boldsymbol{U}\boldsymbol{Op}\boldsymbol{U}^{-1} = \boldsymbol{Op}^{\boldsymbol{DB}}$$

Here ***U*** is the matrix representation of a basis, ***Op*** the matrix representation of an operator in the basis, and superscript ***DB*** indicates the default basis.

The *Default Basis* is the **reference basis** of GAMMA, *i.e.* the matrix representation of this basis is simply the identity matrix.***U*** will convert an operator ***Op***

$$\boldsymbol{U}^{\boldsymbol{DB}} = (\boldsymbol{U}^{\boldsymbol{DB}})^{-1} = \boldsymbol{I}$$

## 5.2 Available Basis Functions

### Basis Algebraic

| | | |
|---|---|---|
| basis | - Constructor | page 120 |
| ~basis | - Destructor | page 123 |
| = | - Assignment | page 118 |

### Basic Functions

| | | |
|---|---|---|
| dim | - Basis Dimension | page 122 |
| size | - Basis Dimension) | page 123 |
| U | - Basis Matrix | page 122 |
| convert | - Transform matrix into new basis | page 120 |
| convert_back | - Transform matrix out of basis. | page 121 |

### Basis Comparison & Test Functions

| | | |
|---|---|---|
| == | - Basis Equality | page 118 |
| != | - Basis Inequality) | page 119 |
| isDefaultBasis | - Check for Default Basis) | page 121 |

### Basis Input & Output Functions

| | | |
|---|---|---|
| << | - Basis Output | page 119 |

# 5.3 Routines

## 5.3.1 =

**Usage:**

```
#include <HSLib/Basis.h>
void operator = (basis& bs);
```

**Description:**

Assignment operator = equates one basis to another basis. Via class *Matrix* we avoid copying unless needed[1].

**Example:**

```
#include <gamma.h>
main()
  {
  basis bs1, bs2;                              // Create two new bases.
  bs2=bs1;                                     // Set basis bs2 equivalent to basis bs1.
  }
```

**Return Value:**

Function is void.

**See Also: basis**

## 5.3.2 ==

**Usage:**

```
#include <HSLib/Basis.h>
int operator == (basis& bs1, basis& bs2);
```

**Description:**

The equality function == compares two bases. Returns TRUE if the two are equivalent and FALSE if not.

**Example:**

```
#include <gamma.h>
main()
  {
  basis bs1,bs2;
  if (bs1==bs2) cout << “Basis 1 is equal to basis 2\n”;
  }
```

**Return Value:**

returns true if the two bases are identical.

**See Also:**

=, !=, basis

[1] For more details see class matrix structure discussions..

### 5.3.3 !=

**Usage:**

```
#include <HSLib/Basis.h>
int operator != (basis& bs1, basis& bs2);
```

**Description:**

The inequality funciton != compares two bases. The function returns FALSE if the two bases are equivalent and TRUE if they are inequivalent.

**Example:**

```
#include <gamma.h>
main()
  {
  basis bs1,bs2;
  if (bs1!=bs2)
  cout << “Basis 1 is not equal to basis 2\n”;
  }
```

**Return Value:**

returns true if the two bases are not identical.

**See Also:**

=, ==, basis

### 5.3.4 <<

**Usage:**

```
#include <HSLib/Basis.h>
ostream& operator<< ( ostream& os, basis& bs);
```

**Description:**

Prints a basis to the ostream os.

**Example:**

```
#include <gamma.h>
main()
  {
  basis bs;
  cout << bs
  }
```

**Return Value:**

returns the modified ostream.

**See Also:**

## 5.3.5 basis

**Usage:**

```
#include <HSLib/Basis.h>
 basis (int dim);
 basis (matrix& mx);
 basis (basis& bs);
```

**Description:**

The purpose of function *basis* is to construct a new basis.With the dimension as the argument it creates the default basis. With a matrix as the argument, the basis is created with a copy of the matrix set as the transformation matrix **U**. With another basis as the argument, it creates the identical basis.

**Example:**

```
#include <gamma.h>
main()
  {
  matrix mx;                              // Declare a matrix mx
  basis bs1(8);                           // Declare a NULL basis of dimension 8, bs1
  basis bs2(mx);                          // Construct basis bs2 with matrix mx
  basis bs3(bs2);                         // Construct basis bs3 equal to bs2
  }
```

**Return Value:**

The newly created basis

**See Also:**

=, ==

## 5.3.6 convert

**Usage:**

```
#include <HSLib/Basis.h>
matrix convert (basis& bs, matrix& mx);
matrix convert (basis& bs1, basis& bs2, matrix& mx);
```

**Description:**

Converts an input matrix (operator) from the default basis into the basis indicated.

$$\boldsymbol{U}^{a\dagger}\boldsymbol{Op}^{DB}(\boldsymbol{U}^{a}) = \boldsymbol{Op}^{a}$$

**Example:**

```
#include <gamma.h>
main()
  {
  matrix mx1,mx2;                         //decare two matrices
  basis bs1,bs2;                          //declare two bases
  mx2 = convert (bs1, mx1);               //mx2 is matrix mx1(default basis) in basis c
  mx2 = convert (bs1, bs2, mx1);          //converts mx1 from basis bs1 to basis bs2
```

**Return Value:**

returns the converted matrix.

**See Also:**

convert_back

## 5.3.7 convert_back

**Usage:**

```
#include <HSLib/Basis.h>
matrix convert_back (basis& a, matrix& b);
```

**Description:**

The function *convert_back* converts a matrix from the basis indicated back into the default basis.

$$\boldsymbol{U}^a \boldsymbol{Op}^a \boldsymbol{U}^{a\dagger} = \boldsymbol{Op}^{DB}$$

**Return Value:**

returns the matrix expressed in the default basis.

**See Also: convert**

## 5.3.8 isDefaultBasis

**Usage:**

```
#include <HSLib/Basis.h>
int isDefaultBasis ();
```

**Description:**

The function *isDefaultBasis* test wether a basis is the default basis or not.

**Example:**

```
#include <gamma.h>
main()
  {
  basis c(8);                                    // create default basis c.
  if (c.isDefaultBasis() ) cout << “c is the default basis.”;
  }
```

**Return Value:**

TRUE if  basis is the default basis.

**See Also: basis**

## 5.3.9 U

**Usage:**

```
#include <HSLib/Basis.h>
matrix U(basis& bs);
```

**Description:**

Function ***U*** returns the transformation matrix, $\boldsymbol{U}^{\boldsymbol{bs}}$, which relates the basis indicated, ***bs***, to the default basis, ***DB.***

$$U^a Op^a U^{a\dagger} = Op^{DB}$$

where ***Op*** is any operator.

**Example:**

```
#include <gamma.h>
main()
  {
  matrix a,b;
  basis bs(a);                          // Create basis bs with transformation matrix a.
  b = bs.U();                           // Now matrix b should be equal to matrix a
  }
```

**Return Value:**

returns the conversion matrix.

**See Also: basis**

## 5.3.10 dim

**Usage:**

```
#include <HSLib/Basis.h>
int basis::dim();
```

**Description:**

The function ***dim*** returns the dimenstion of the basis trasformation matrix, $U$.

**Example:**

```
#include <gamma.h>
main()
  {
  basis bs;                             // declare a basis bs
  int i;                                // declare an integer i
  i = bs.dim();                         // set integer i to the dimension of basis bs
  }
```

**Return Value:**

returns the integer value of the basis dimension.

**See Also:**

## 5.3.11 size

**Usage:**

```
#include <HSLib/Basis.h>
int basis::size();
```

**Description:**

The function ***size*** is equivalent to the function ***dim*** both in useage and in its return value.

**Example:**

```
#include <gamma.h>
main()
  {
  basis bs;                                    // declare a basis bs
  int i;                                       // declare an integer i
  i = bs.size();                               // set integer i to the dimension of basis bs
  }
```

**Return Value:**

returns the integer value of the basis dimension.

**See Also:**

## 5.3.12 ~basis

**Usage:**

```
#include <HSLib/Basis.h>
 ~basis ();
```

**Description:**

Destroys a basis.

**Return Value:**

Nothing

**See Also:**

basis

## 5.4 Description

A basis merely maintains a transformation matrix ***U*** (see class ***matrix***). Usually ***U*** is associated with the eigenbasis of some operator ***Op*** (see class ***operator***) but any basis can act as an independent quantity and be used to manipulate arbitrary matrices or operators if desired.

By definition, the transformation matrix ***U*** will convert an operator ***Op*** (or any matrix) between the basis itself and the **GAMMA default basis**, as given by the similarity transformation equation

$$\boldsymbol{U}^{a}\boldsymbol{Op}^{a}(\boldsymbol{U}^{a})^{-1} = \boldsymbol{Op}^{DB}. \tag{5-1}$$

Here ***U*** is the matrix representation of a basis ***a*** indicated by superscript ***a***, ***Op*** is the matrix representation of an operator in that basis, superscript ***-1*** is used to indicate the inverse, and superscript ***DB*** indicates the default basis[1].

The *default basis* is the **reference basis** of GAMMA, *i.e.* the matrix representation of this basis is simply the identity matrix.

$$(\boldsymbol{U}^{DB})^{-1} = \boldsymbol{U}^{DB} = \boldsymbol{I}. \tag{5-2}$$

Typically, several operators will have representations in the default basis and therefore be associated with the same basis $\boldsymbol{U}^{\boldsymbol{DB}}$. class ***matrix*** takes care that the transformation matrices are only stored once. An operator or matrix in the default basis may be placed into a base of choice once the basis (or base transformation matrix ***U***) has been determined. This is of course given by the inverse of (5-1).

$$\boldsymbol{Op}^{a} = (\boldsymbol{U}^{a})^{-1}\boldsymbol{Op}^{DB}\boldsymbol{U}^{a} \tag{5-3}$$

The *eigenbasis* of an operator is the basis in which the operator (not the transformation matrix ***U***) is represented by a diagonal matrix. Using superscript ***EB*** to indicate the eigenbasis and $\Lambda$ to indicate the diagonal representation of operator ***Op***, the relationship between the eigenbasis (of ***Op***) and the default basis is

$$\boldsymbol{U}^{EB}\boldsymbol{Op}^{EB}(\boldsymbol{U}^{EB})^{-1} = \boldsymbol{U}^{EB}\Lambda^{EB}(\boldsymbol{U}^{EB})^{-1} = \boldsymbol{Op}^{DB}. \tag{5-4}$$

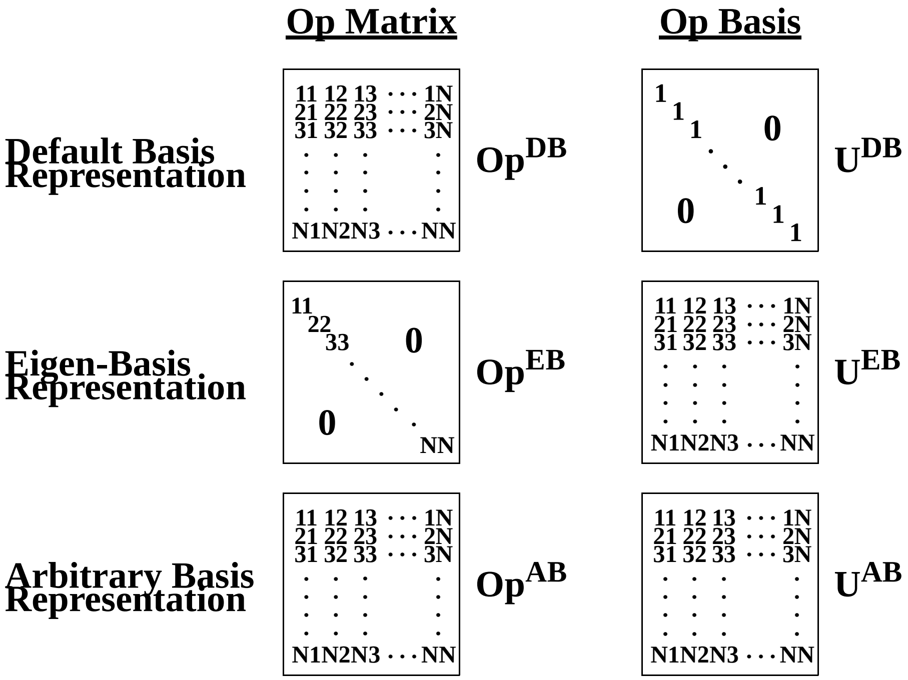
The above equation implies that the eigenbasis matrix, $\boldsymbol{U}^{\boldsymbol{DB}}$, is obtained from diagonalization of the operator $\boldsymbol{Op}^{\boldsymbol{DB}}$ in the default basis.

$$\boldsymbol{Op}^{EB} = \Lambda^{EB} = (\boldsymbol{U}^{EB})^{-1}\boldsymbol{Op}^{DB}\boldsymbol{U}^{EB} \tag{5-5}$$

Obviously, each operator can be associated with several bases - the default basis, the eigenbasis, and any number of arbitrary bases. The diagram below depicts an operator with three representa-

[1] Often the basis array is unitary and its inverse is equivalent to its adjoint. This is particularly true in many magnetic resonance simulations where the operator is Hermitian and, as a result, its bases are unitary.

tions using superscript ***AB*** to indicate an arbitrary basis.



As more operators and bases are set up, the internal structure of each basis remains the same, becomes somewhat more complicated but GAMMA attempts to maintain the minimum amount of array allocation for the various bases used.

## 5.5 Implementation

### 5.5.1 Data Structure

Basis is a private derived class from matrix and doesn't contain any of its own data structures.

### 5.5.2 Algorithms

All operations are based upon class matrix. It is assumed that the transformation matrix is unitary, but it is not tested. This should be done in a later stage of the project.

# 6 Class Operator (gen_op)

## 6.1 Overview

The *Class OPERATOR* defines all the necessary attributes of a quantum mechanical operator, **Op**. The essential components of every operator are a matrix (*See Class MATRIX*) and a basis (*See Class BASIS*). This matrix-basis pairing is referred to as an operator representation. There may be several different representations of the same **Op** needed for a calculation, each containing equivalent information about **Op** but existing in a unique basis. GAMMA will automatically compute, maintain, and efficiently utilize these representations so that the user can have minimum concern over their existence. The representation of **Op** which was last utilized in a computation is referred to as the working basis represenation and the associated basis called the working basis.

Within *Class OPERATOR* are also specifiations of operator properties (dimension, ...), operator algebras (+, *,...), and definitions of all available operator functions (exp, trace, ...). Functions are also provided which allow the user direct access into the different represenations of each **Op**.

## 6.2 Available Operator Functions

### Operator Algebraic Functions

| | | | |
|---|---|---|---|
| gen_op | - Constructor: | Op, Op(mx), Op(mx1, mx2),<br>Op(mx, bs), Op(Op1). | page 128 |

### Operator - Operator Functions

| | | | |
|---|---|---|---|
| + | - Addition: | Op1 + Op2 | page 129 |
| += | - Unary Addition: | Op1 += Op2 | page 130 |
| - | - Subtraction: | Op1 - Op2 | page 130 |
| -= | - Unary Subtraction: | Op1 -= Op2 | page 131 |
| -= | - Negation: | -Op | page 130 |
| * | - Multiplication: | Op1 * Op2 | page 132 |
| *= | - Unary Multiplication: | Op1*= Op2 (Op1 = Op1 * Op2) | page 133 |
| &= | - Reverse Unary Multiplication: | Op1&= Op2 (Op1 = Op2 * Op1) | page 134 |
| = | - Assignment: | Op1 = Op2 | page 135 |

### Operator - Matrix Functions

| | | | |
|---|---|---|---|
| + | - Addition: | Op + mx, mx+Op | page 129 |
| += | - Unary Addition: | Op+ = mx | page 130 |
| - | - Subtraction: | Op - mx, mx - Op | page 130 |
| -= | - Unary Subtraction: | Op -= mx. | page 131 |
| * | - Multiplication: | Op*mx, mx*Op | page 132 |
| *= | - Unary Multiplication: | Op *=mx (Op = Op * mx) | page 133 |
| &= | - Reverse Unary Multiplication: | Op &= Op2 (Op = mx * Op) | page 134 |

| | | | |
|---|---|---|---|
| = | - Assignment: | Op = mx. | page 135 |

### Operator - Scalar Functions

| | | | |
|---|---|---|---|
| * | - Multiplication: | Op*z, z*Op | page 132 |
| *= | - Unary Multiplication: | Op *=z | page 133 |
| / | - Division: | Op/ z | page 135 |
| /= | - Unary Division: | Op /= z | page 135 |

### Complex Operator Functions

| | | | |
|---|---|---|---|
| det | - Operator determinant: | det(Op) | page 137. |
| trace | - Operator trace: | trace(Op), trace(Op1, Op2) | page 137 |
| proj | - Operator projection: | Op1.proj(Op2, norm) | page 137 |
| size | - Operator size: | Op.size() | page 138 |
| dim | - Operator dimension: | Op.dim() | page 138. |
| exp | - Operator exponential: | exp(Op) | page 139 |
| sim_trans | - Operator similarity transform: | Op1.sim_trans(Op2) | page 139 |
| sim_trans_ip | - Operator sim. trans. in place: | Op1.sim_transi_ip(Op2). | page 139 |
| adjoint | - Operator Hermitian adjoint: | adjoint(Op) | page 139 |
| eigvals | - Operator eigenvalues: | Op.eigvals; | page 139 |

### Operator Internal Access

| | | | |
|---|---|---|---|
| exp | - Retrieve operator matrix. | Op.size() | page 139. |
| exp | - Assign operator matrix. | Op.size() | page 139. |
| exp | - Retrieve operator basis. | Op.size() | page 139. |
| put_basis | - Assign operator basis. | Op.size() | page 139. |
| () | - Retrieve operator element: | z = LOp(3,2) | page 139. |
| get | - Retrieve operator element: | Op.size() | page 139 |
| put | - Assign operator element: | Op.size() | page 139 |
| exists | - Operator existence: | Op.exists; | page 139 |

### Basis Manipulations

| | | | |
|---|---|---|---|
| set_DBR | - Put operator into its default basis: | Op.size() | page 143. |
| set_EBR | - Put operator into its eigenbasis: | Op.size() | page 143. |
| Op_base | - Put operator into a specific basis: | Op.size() | page 143 |
| test_EBR | - Test if operator in its eigenbasis: | Op.size() | page 143 |
| status | - Output operator rep.status | Op.size() | page 143 |
| set_only_WBR | - Remove all reps but current: | Op.size() | page 143 |

### Representation Manipulations

| | | | |
|---|---|---|---|
| limit_reps | - Limit number of operator reps: | Op.size() | page 146 |
| Op_priority | - Assin a priority level to a rep: | Op.size() | page 146. |
| set_limits | - Specify number of reps: | Op.size() | page 147. |

### Operator I/O Functions

| | | | |
|---|---|---|---|
| << | - Send operator to output stream:. | Op.size() | page 146 |

# 6.3 Arithmetic Operators

## 6.3.1 gen_op

### Usage:

```
#include <HSLib/GenOp.h>
gen_op ( )
gen_op ( matrix& mx )
gen_op ( matrix& mx, basis& bs )
gen_op ( gen_op& Op )
```

### Description:

The function *gen_op* is used to create an operator quantity.

1. gen_op() - sets up an empty operator which can be explicitly specified later.
2. gen_op(matrix) - sets up an operator with the matrix **mx** in the argument assumed to be the operator representation in the default basis. The specified matrix must be a square array.
3. gen_op(matrix, basis) - With a matrix **mx** and a basis **bs** as arguments, the function sets up an operator with matrix representation **mx** in the basis **bs**. Again, **mx** must be a square matrix and its dimension must match that of the basis **bs**. The basis must relate properly to the default basis, meaning that the basis transformation matrix can be used to transform **mx** into the default basis (see *Class BASIS*).
4. gen_op(operator) - Finally, one may produce an operator from another operator. The new operator will then be equivalent to the *current representation* of the input operator **Op**.

### Return Value:

Creates a new operator which may be subsequently used with all defined operator functions.

### Example(s):

```
#include <gamma.h>
main()
  {
  matrix mx;
  basis bs;
  gen_op Op;                          // produces an empty operator Op.
  gen_op Op1(mx);                     // produces Op1 with matrix mx in the default basis.
  gen_op Op2(mx, bs);                 // produces Op2 with matrix mx in the basis bs.
  gen_op Op3(Op2);                    // produces Op3 with1 representation equal to Op2.
  }
```

### See Also:

None

## 6.3.2 +

### Usage:

```
#include <HSLib/GenOp.h>
gen_op operator + (gen_op& Op1, gen_op& Op2)
gen_op operator + (gen_op& Op, matrix& mx)
gen_op operator + (matrix& mx, gen_op& Op)
void operator + (gen_op& Op)
```

### Description:

This allows for the addition of two operators, **Op1** + **Op2** and the addition of an operator with a matrix, **Op1** + **mx**.

1. Op1 + Op2 - Definition of the addition of two operators **Op1** and **Op2**. A check is made to insure that both operators are in the same basis. If this is not true, operator **Op2** is transformed into the basis of operator **Op1** prior to the addition, thus insuring that the addition *produces a result in (and only in) the same basis of* **Op1**.
2. Op + mx - Definition of the addition of a matrix mx to an operator **Op**. The matrix mx is assumed to be a matrix in the default basis and the addition takes place in the default basis. Operator **Op** is first placed in the default basis, the addition takes place, and then the result is *new operator in the default basis*.
3. mx + Op - Definition of the addition of an operator **Op** to a matrix mx. The result is equivalent to the previous addition, it produces a *new operator in the default basis*.
4. + Op - Unary plus added to be consistant with negation operation. Returns **Op** unchanged.

Addition inherently **works on only one operator representation**, *i.e.* the formula **Op3** = **Op1** + **Op2** produces the operator **Op3** in a single representation ( in the basis of operator **Op1**) reguardless of how many stored representations of **Op1** and/or **Op2** exist. Since GAMMA will transform **Op3** into any needed basis automatically, this should present no limitations while keeping computation time and memory useage down. One should keep in mind that a consequence of this is that the operation **Op1** = **Op1** + **Op2** will set any current representations of operator **Op1** to zero except the result representation. Therefore, if one has several representations of **Op1** and **Op2** and desires **Op3** = **Op1** + **Op2** to include all of these representations, use the provided function *add_operators*. This function will produce all representations of **Op3** for the bases of **Op1** by operator element additions rather than similarity transformations. In this case, assuming the user actually accesses all the representations of **Op3**, this latter method will save on computation time as well.

### Return Value:

A new operator which exists in an appropriate representation.

### Example(s):

```
#include <gamma.h>
main()
  {
  matrix mx;
  gen_op Op1,Op2,Op3;                // define three operators Op1,Op2, and Op3.
  Op3 = Op1 + Op2;                   // Op3 set to the sum Op1 + Op2. Only in Op1 WB .
  Op3 = mx + Op1;                    // Op3 set to the sum Op1 + matrix mx. Only in DB.
  Op3 = Op1 + mx;                    // same as the previous line.
  }
```

### See Also:

-

## 6.3.3 +=

### Usage:

#include <HSLib/GenOp.h>

gen_op operator += (gen_op& Op)

### Description:

This allows for the addition of two operators of the type **Op1 = Op1** + **Op2**. When both operators exist, a check is made to insure that both operators are in the same basis. If necessary, operator **Op2** is transformed into the basis of operator **Op1** prior to the addition, thus insuring that the addition *produces a result in (and only in) the same basis of* **Op1**. Use of this operator is more computationally efficient that the two step operation.That is, the statement **Op1** += **Op2**; is preferred over the statement **Op1** = **Op1** + **Op2**;. In the event that **Op1** does not exist, this function acts like the assignment operator to produce **Op1 = Op2**. In the event that **Op2** does not exist, this function has no effect on **Op1**

### Return Value:

A new operator which exists in an appropriate representation.

### Example(s):

```
#include <gamma.h>
main()
  {
  matrix mx;
  gen_op Op1,Op2;                          // define two operators Op1 and Op2.
  Op1 += Op2;                              // Op1 is sum of Op1 + Op2. Only in the WB of Op1.
  }
```

### See Also:

+, -, -=

## 6.3.4 -

### Usage:

#include <HSLib/GenOp.h>

gen_op operator - (gen_op& Op1, gen_op& Op2)

gen_op operator - (gen_op& Op, matrix& mx)

gen_op operator - (matrix& mx, gen_op& Op)

gen_op operator- (gen_op& Op)

### Description:

This allows for the subtraction of two operators **Op1** and **Op2**, for the subtraction of a scalar from an operator, and for the negation of an operator. This allows for the addition of two operators, **Op1** + **Op2** and the addition of an operator with a matrix, **Op** + mx.

1. Op1 - Op2 - Definition of the subtraction of two operators **Op1** and **Op2**. A check is made to insure that both operators are in the same basis. If this is not true, operator **Op2** is placed into the basis of operator **Op1** prior to the subtraction, thus insuring that the subtraction *produces a result in (and only in) the same basis of* **Op1**.
2. Op - mx - Definition of the subtraction of a matrix mx from an operator **Op**. The matrix mx is assumed

to be a matrix in the default basis and the subtraction takes place in the default basis. Operator **Op** is first placed in the default basis, the matrix mx is then subtracted from it , and then the result is *new operator in the default basis*.

3. mx - Op - Definition of the subtraction of an operator **Op** from a matrix mx. The result is equivalent to the negative of the previous subtraction, it produces a *new operator in the default basis*.
4. - Op - Definition of the negation of operator **Op**. The result is an operator in (and only in) the working basis of **Op** which is -1.0 * **Op**.

Subtraction, like addition, inherently **works on only one operator representation**, *i.e.* the formula **Op3** = **Op1** - **Op2** produces the operator **Op3** in a single representation ( in the basis of operator **Op1**) reguardless of how many stored representations of **Op1** and/or **Op2** exist. Since GAMMA will transform **Op3** into any needed basis automatically, this should present no limitations while keeping computation time and memory useage down. One should keep in mind that a consequence of this is that the operation **Op1** = **Op1** - **Op2** will set any current representations of operator Op1 to zero except the result. Therefore, if one has several representations of **Op1** and **Op2** and desires **Op3** = **Op1** - **Op2** to include all of these representations, use the provided function *subtract_operators*. This function will produce all representations of **Op3** for the bases of **Op1** by operator element subtractions rather than similarity transformations. In this case, assuming the user actually accesses all the representations of**Op3**, this latter method will save on computation time as well.

### Return Value:

For the subtraction of two operators, an operator which is the difference of operators **Op1** and **Op2** is returned in the basis of operator **Op1**. For the subtraction of a matrix from an operator, an operator which has the matrix subtracted is returned in the default basis.. The negation returns an operator whose elements are the negatives of the input operator in the working basis of the input operator.

### Example(s):

```
#include <gamma.h>
main()
  {
  matrix mx;
  gen_op Op1,Op2,Op3;                    // define three operators Op1,Op2, and Op3.
  Op3 = Op1 - Op2;                       // Op3 set Op1 - Op2. Only in the WB of Op1.
  Op3 = Op1 -mx ;                        // Op3 set to Op1 minus matrix mx. Only in DB.
  Op3 = mx - Op1 ;                       // same as -(Op1 - mx). Only in default basis of Op1.
  Op3 = -Op1 ;                           // Op3 set to negative Op1. Only in the WB of Op1.
  }
```

**See Also:+, *, /**

## 6.3.5 -=

### Usage:

#include <HSLib/GenOp.h>

gen_op operator -= (gen_op& Op)

### Description:

This allows for the subtraction of two operators of the type **Op1 = Op1 - Op2**. A check is made to insure that both operators are in the same basis. If this is not true, operator **Op2** is transformed into the basis of operator **Op1** prior to the addition, thus insuring that the subtraction *produces a result in (and only in) the same basis of* **Op1**. Use of this operator is more computationally efficient that the two step operation.That is, the statement **Op1** -= **Op2**; is preferred over the statement **Op1** = **Op1** - **Op2**;.

### Return Value:

A modifoed operator which exists in an appropriate representation.

### Example:

```
#include <gamma.h>
main()
  {
  gen_op Op1,Op2;                     // define two operators Op1 and Op2.
  Op1 -= Op2;                         // Op1 set to Op1 - Op2. Only in WB of Op1.
  }
```

### See Also:

+, +=, -

## 6.3.6 *

### Usage:

```
#include <HSLib/GenOp.h>
gen_op operator * (gen_op& Op1, gen_op& Op2)
gen_op operator * (gen_op& Op, matrix& mx)
gen_op operator * (matrix& mx, gen_op& Op)
gen_op operator * (complex& z, gen_op& Op)
gen_op operator * (gen_op& Op, complex& z)
```

### Description:

This allows for the multiplication of two operators **Op1** and **Op2**, the multiplication of an operator and a matrix, and for the multiplication of a scalar and an operator.

1. For the multiplication of two operators, a check is made to insure that both operators are in the same basis. If this is not true, operator **Op2** is transformed into the basis of operator **Op1** prior to the multiplication, thus insuring that the subtraction *produces a result in the same basis of* **Op1**. Here, the order of the operators can make a difference in the result.
2. For the multiplication of an operator times a matrix, an operator is produced. It is assumed that the matrix is a quantity in the default basis so that operator **Op** is changed into the default basis before the multiplication.
3. The multiplication of a matrix time an operator also produces an operator. The treatment is similar to the previous useage except the ordering can make a difference.
4. For the multiplication of a scalar times an operator, the complex scalar z is multiplied into each element of **Op** to produce an operator in the same basis of **Op.**
5. The multiplication of an operator times a scalar produces the same result as multiplication of a scalar times an operator.

Multiplication, like addition and subtraction, inherently **works on only one operator representation**, *i.e.* the formula **Op3** = **Op1** * **Op2** produces the operator **Op3** in a single representation ( in the basis of operator **Op1**) reguardless of how many stored representations of **Op1** and/or **Op2** exist. Since GAMMA will transform **Op3** into any needed basis automatically, this should present no limitations while keeping computation time and memory useage down. One should keep in mind that a consequence of this is that the operation **Op1** = **Op1** * **Op2** will set any current representations of operator **Op1** to zero except the result. Therefore, if one has several representations of **Op1** and **Op2** and desires **Op3** = **Op1** * **Op2** to include all of these representations, use the provided function *multiply_operators*. This function will produce all representations of **Op3** for the bases of **Op1** by operator multiplications rather than similarity transformations (two matrix multiplications). In this case, assuming the user actually accesses all the representations of **Op3**, this latter method will save on computation time as well.

### Return Value:

none.

### Example(s):

```
#include <gamma.h>
main()
  {
  complex z;
  gen_op Op1,Op2,Op3;                  // define three operators Op1,Op2, and Op3.
  Op3 = Op1 * Op2;                     // Op3 set to Op1 x Op2. Only in WB of Op1.
  Op3 = Op2 * Op1;                     // may not be same as previous line, order matters.
  Op3 = Op1* z ;                       // Op3 set to Op1 with all elements multiplied by z.
  Op3 = z * Op1 ;                      // same as previous line. Only in the WB of Op1.
  }
```

### See Also:

*=, +, -, /

## 6.3.7 *=

### Usage:

#include <HSLib/GenOp.h>
gen_op operator *= (gen_op& Op1)
gen_op operator *= (matrix& mx)
gen_op operator *= (complex& z)

### Description:

1. **Op2 *= Op1 -** For the multiplication of two operators, a check is made to insure that both operators are in the same basis. If this is not true, operator **Op1** is transformed into the basis of operator **Op2** prior to the multiplication, thus insuring that the subtraction *produces a result in the same basis of* **Op2**. Here, the order of the operators can make a difference in the result.
2. **Op *= mx -** For the multiplication of an operator times a matrix, an operator is produced. It is assumed that the matrix is a quantity in the default basis so that operator **Op** is changed into the default basis before the multiplication.
3. **Op *=** z **-** The multiplication of a operator times a scalar (complex) also produces an operator. This operation *produces a result exclusively in the original working basis of* **Op**.

This allows for the multiplication of two operators of the type **Op1 = Op1 * Op2**. A check is made to insure that both operators are in the same basis. If this is not true, operator **Op2** is transformed into the basis of operator **Op1** prior to the addition, thus insuring that the addition *produces a result in (and only in) the same basis of* **Op1**. Use of this operator is more computationally efficient that the two step operation.That is, the statement **Op1** *= **Op2**; is preferred over the statement **Op1** = **Op1** * **Op2**;.

Addition inherently **works on only one operator representation**, *i.e.* the formula **Op1** *= **Op2** produces the operator **Op1** in a single representation ( in the working basis of operator **Op1**) reguardless of how many stored representations of **Op1** and/or **Op2** exist. A consequence of this is that the operation **Op1** *= **Op2** will set any current representations of operator **Op1** to zero except the result representation. Therefore, if one has several representations of **Op1** and **Op2** and desires **Op1** *= **Op2** to include all of these representations, use the provided function *add_operators*.

### Return Value:

A new operator which exists in an appropriate representation.

### Example(s):

```
#include <gamma.h>
main()
  {
  matrix mx;
  gen_op Op1,Op2;                          // define two operators Op1 and Op2.
  Op1 *= Op2;                              // Op1 set to Op1 * Op2. Only in Op1 working basis.
  }
```

### See Also:

&=, -

## 6.3.8 &=

### Usage:

#include <HSLib/GenOp.h>
gen_op operator &= (gen_op& Op1)

### Description:

This allows for the multiplication of two operators of the type **Op1 = Op2*Op1**. A check is made to insure that both operators are in the same basis. If this is not true, operator **Op2** is transformed into the basis of operator **Op1** prior to the multiplication, thus insuring that the multiplication *produces a result in (and only in) the same basis of* **Op1**. Use of this operator is more computationally efficient that the two step operation.That is, the statement **Op1** &= **Op2**; is preferred over the statement **Op1** = **Op2 * Op1**;.

Multipliation inherently **works on only one operator representation**, *i.e.* the formula **Op1** &= **Op2** produces the operator **Op1** in a single representation (in the working basis of operator **Op1**) reguardless of how many stored representations of **Op1** and/or **Op2** exist. A consequence of this is that the operation **Op1** &= **Op2** will set any current representations of operator **Op1** to zero except the result representation.

### Return Value:

None, the function is void.

### Example(s):

```
#include <gamma.h>
main()
  {
  matrix mx;
  gen_op Op1,Op2;                          // define two operators Op1 and Op2.
  Op1 &= Op2;                              // Op1 set to Op2 * Op1. Only in the WB of Op1.
  }
```

### See Also:

*=, -, add_operators

### 6.3.9 /

**Usage:**

```
#include <HSLib/GenOp.h>
gen_op operator / (complex& z, gen_op& Op)
gen_op operator / (gen_op& Op, complex& z)
```

**Description :**

**Return Value:**

### 6.3.10 /=

**Usage:**

```
#include <HSLib/GenOp.h>
gen_op operator /= (complex& z)
```

**Description :**

**Return Value:**

### 6.3.11 =

**Usage:**

```
#include <HSLib/GenOp.h>
gen_op operator = (gen_op& Op)
gen_op operator = (matrix& mx)
```

**Description:**

This allows for the ability to equate an operator to another operator or equate an operator to a matrix.

1. For the equating of two operators, **Op1** = **Op2**, operator **Op1** is set equalt to operator **Op2** in (exclusively) the working basis of operator **Op2**.
2. For the equating of an operator to a matrix, **Op** = mx, the matrix mx is taken to represent the operator **Op** in the default basis. Any other current representations of operator **Op** will be deleted.

The equality **works on only one operator representation**, *i.e.* the formula **Op1** = **Op2** produces the operator **Op1** in a single representation ( in the basis of operator **Op2**) reguardless of how many stored representations of **Op2** and/exist. Since GAMMA will transform **Op1** into any needed basis automatically, this should present no limitations while keeping computation time and memory useage down. If one has several representations of **Op2** and desires **Op1** = **Op2** to include all of these representations, use the provided function *equate_operators*. This function will produce all representations of **Op1** for the bases of **Op2**.

**Return Value:**

none.

### Example(s):

```
#include <gamma.h>
main()
  {
  matrix mx;
  gen_op Op1,Op2;                         // define two operators Op1 and Op2.
  Op1 = Op2;                              // Op1 set equal to Op2. Only in working basis of Op2.
  Op1 = mx;                               // Op1 set to matrix mx assumed in the default basis.
  }
```

### See Also:

none

# 6.4 Complex Functions

## 6.4.1 det

### Usage:

#include <HSLib/GenOp.h>
complex operator det(gen_op& Op)

### Description :

Computes the determinant of an operator.

### Return Value:

## 6.4.2 trace

### Usage:

#include <HSLib/GenOp.h>
complex trace(gen_op& Op)
complex trace(gen_op& Op1, gen_op& Op2)

### Description :

Computes the trace of an operator. Although calculated in the current working basis of the operator, trace should be invarient for all representations of the same operator.

1. Computes the trace of the general operator **Op** via summation over its diagonal elements.
2. Computes the trace of the general operator (**Op1** * **Op2**), *i.e.* the trace of the operator resulting from the multiplicaton of the two input operators. Use of this function is more efficient that the analogous two step process, multiplication followed by trace, because the full operator multiplication of **Op1** and **Op2**is not performed. Rather use is made of the following equation, where the trace is obtained with $n^2$ multiplication steps where n is the dimension of the operators.

$$Tr\{\boldsymbol{Op}1 \bullet \boldsymbol{Op}2\} = \sum_i \langle i|\boldsymbol{Op}1 \bullet \boldsymbol{Op}2|i\rangle = \sum_i \sum_i \langle i|\boldsymbol{Op}1|j\rangle \langle j|\boldsymbol{Op}2|i\rangle \qquad (9\text{-}1)$$

### Return Value:

Complex number.

### Example(s):

```
#include <gamma.h>
main()
  {
  gen_op Op1, Op2, Op3;                 // define two operators Op1 and Op2.
  complex z;
  z = trace(Op1);                       // Set z to the trace of operator Op1.
  z = trace(Op1, Op2);                  // Set z to the trace of (Op1 * Op2).
  // Op3 = Op1 * Op2;                   // These two steps are less efficient than
  // z = trace(Op3);                    // using z = trace(Op1 * Op2) !!
  }
```

## 6.4.3 proj

### Usage:

#include <HSLib/GenOp.h>
complex trace(gen_op& Op)
complex trace(gen_op& Op1, gen_op& Op2)

### Description :

Computes the trace of an operator. Although calculated in the current working basis of the operator, trace should be invarient for all representations of the same operator.

1. Computes the trace of the general operator **Op** via summation over its diagonal elements.
2. Computes the trace of the general operator (**Op1** * **Op2**), *i.e.* the trace of the operator resulting from the multiplicaton of the two input operators. Use of this function is more efficient that the analogous two step process, multiplication followed by trace, because the full operator multiplication of **Op1** and **Op2**is not performed. Rather use is made of the following equation, where the trace is obtained with n$^2$ multiplication steps where n is the dimension of the operators.

$$Tr\{\boldsymbol{Op}1 \bullet \boldsymbol{Op}2\} = \sum_i \langle i|\boldsymbol{Op}1 \bullet \boldsymbol{Op}2|i\rangle = \sum_i \sum_i \langle i|\boldsymbol{Op}1|j\rangle \langle j|\boldsymbol{Op}2|i\rangle \qquad (9\text{-}1)$$

### Return Value:

Complex number.

### Example(s):

```
#include <gamma.h>
main()
  {
  gen_op Op1, Op2, Op3;                    // define two operators Op1 and Op2.
  complex z;
  z = trace(Op1);                          // Set z to the trace of operator Op1.
  z = trace(Op1, Op2);                     // Set z to the trace of (Op1 * Op2).
  // Op3 = Op1 * Op2;                      // These two steps are less efficient than
  // z = trace(Op3);                       // using z = trace(Op1 * Op2) !!
  }
```

## 6.4.4 size

### Usage:

#include <HSLib/GenOp.h>
int gen_op::size()

### Description :

The function *size* works identically to the general operator member function *dim*. See its description.

### 6.4.5 dim

**Usage:**

```
#include <HSLib/GenOp.h>
int dim(gen_op& Op)
```

**Description :**

The function *dim* allows the user to access the dimension of the operator given in the argument.

**Return Value:**

Integer.

**Example(s):**

```
#include <gamma.h>
main()
  {
  gen_op Op;                          // define an operator Op.
  int size;                           // define an integer.
  size = trace(Op);                   // Set the integer to the dimension of the operator Op.
  }
```

### 6.4.6 exp

**Usage:**

```
#include <HSLib/GenOp.h>
gen_op exp (gen_op& Op)
```

**Description :**

Computes the exponential of an operator. The output operator is in the working basis of the input operator. The computation is done in the eigenbasis of the input operator.

### 6.4.7 sim_trans

**Usage:**

```
#include <HSLib/GenOp.h>
gen_op gen_op::sim_trans(gen_op& Op)
```

**Description :**

### 6.4.8 sim_trans_ip

**Usage:**

```
#include <HSLib/GenOp.h>
gen_op gen_op::sim_trans_ip(gen_op& Op)
```

**Description :**

### 6.4.9 adjoint

**Usage:**

#include <HSLib/GenOp.h>
gen_op operator exp (gen_op& Op)

**Description :**

Computes the exponential of an operator. The output operator is in the working basis of the input operator. The computation is done in the eigenbasis of the input operator.

### 6.4.10 eigvals

**Usage:**

#include <HSLib/GenOp.h>
row_vector gen_op::eigvals(gen_op& Op)
void gen_op::(double* vx);

**Description :**

# 6.5 Operator Internal Access

## 6.5.1 get_mx

**Usage:**

```
#include <HSLib/GenOp.h>
matrix gen_op::get_,mx();
```

**Description :**

## 6.5.2 put_mx

**Usage:**

```
#include <HSLib/GenOp.h>
void gen_op::put_,mx();
```

**Description :**

## 6.5.3 get_basis

**Usage:**

```
#include <HSLib/GenOp.h>
basis gen_op::get_,basis();
```

**Description :**

## 6.5.4 put_basis

**Usage:**

```
#include <HSLib/GenOp.h>
void gen_op::put_,basis();
```

**Description :**

## 6.5.5 ( )

**Usage:**

```
#include <HSLib/GenOp.h>
complex operator () (int row, int col);
```

**Description :**

### 6.5.6 put

**Usage:**

```
#include <HSLib/GenOp.h>
void gen_op::put(complex &z, int row, int col);
```

**Description :**

### 6.5.7 get

**Usage:**

```
#include <HSLib/GenOp.h>
complex gen_op::get(int row, int col);
```

**Description :**

### 6.5.8 exists

**Usage:**

```
#include <HSLib/GenOp.h>
int gen_op::exists();
```

**Description :**

# 6.6 Basis Manipulations

## 6.6.1 set_DBR

**Usage:**

```
#include <HSLib/GenOp.h>
void set_DBR (gen_op& Op)
```

**Description :**

The function *set_DBR* insures that **Op** is currently in its default basis representation. If the default basis representaion of **Op** is not internally maintained it will be computed by similarity transformation then stored (within the confines of imposed limits set for the representaions of **Op**). **Op** will then have its working basis set to its default basis.

**Return Value:**

none

**Example(s):**

```
#include <gamma.h>
main()
  {
  gen_op Op1,Op2;                          // define two operators Op1 and Op2.
  }
```

## 6.6.2 set_EBR

**Usage:**

```
#include <HSLib/GenOp.h>
void set_EBR (gen_op& Op)
```

**Description :**

The function *set_EBR* insures that **Op** is currently in its eigenbasis representation. If the eigenbasis representaion of **Op** is not internally maintained it will be computed by matrix diagonalization then stored (within the confines of imposed limits set for the representaions of **Op**). **Op** will then have its working basis set to its eigenbasis.

**Return Value:**

none

**Example(s):**

```
#include <gamma.h>
  main()
  {
  gen_op Op1,Op2;                          // define two operators Op1 and Op2.
  }
```

## 6.6.3 Op_base

**Usage:**

```
#include <HSLib/GenOp.h>
void Op_base (gen_op& Op)
void Op_base (basis& bs)
```

**Description :**

The function *set_EBR* insures that **Op** is currently in its eigenbasis representation. If the eigenbasis representaion of **Op** is not internally maintained it will be computed by matrix diagonalization then stored (within the confines of imposed limits set for the representaions of **Op**). **Op** will then have its working basis set to its eigenbasis.

**Return Value:**

none

**Example(s):**

```
#include <gamma.h>
main()
  {
  gen_op Op1,Op2;                              // define two operators Op1 and Op2.
  }
```

## 6.6.4 test_EBR

**Usage:**

```
#include <HSLib/GenOp.h>
int gen_op::test_EBR();
```

**Description :**

The function *set_EBR* insures that **Op** is currently in its eigenbasis representation. If the eigenbasis representaion of **Op** is not internally maintained it will be computed by matrix diagonalization then stored (within the confines of imposed limits set for the representaions of **Op**). **Op** will then have its working basis set to its eigenbasis.

**Return Value:**

none

**Example(s):**

```
#include <gamma.h>
main()
{
gen_op Op1,Op2;                              // define two operators Op1 and Op2.
}
```

## 6.6.5 status

**Usage:**

```
#include <HSLib/GenOp.h>
void gen_op::status();
```

**Description :**

The function *status* writes to standard output the current status of **Op**. First the current number of operator representations is output. Then, each representations priority, type (WBR, EBR, DBR, IBR), # of matrix references, and # of basis representations is output. This function can be useful in following the memory use and efficiency of a GAMMA program.

**Return Value:**

none

**Example(s):**

```
#include <gamma.h>
main()
  {
  gen_op Op;                                          // define operator Op.
  Op.status();                                        // output the current status of Op.
  }
```

## 6.6.6 set_only_WBR()

**Usage:**

#include <HSLib/GenOp.h>
void gen_op::set_only_WBR ()

**Description :**

The function *set_EBR* insures that **Op** is currently in its eigenbasis representation. If the eigenbasis representaion of **Op** is not internally maintained it will be computed by matrix diagonalization then stored (within the confines of imposed limits set for the representaions of **Op**). **Op** will then have its working basis set to its eigenbasis.

**Return Value:**

none

**Example(s):**

```
#include <gamma.h>
main()
  {
  gen_op Op1,Op2;                                     // define two operators Op1 and Op2.
  }
```

# 6.7 Representation Manipulations

## 6.7.1 limit_reps

**Usage:**

#include <HSLib/GenOp.h>
void limit_reps(gen_op& Op, int i)

**Description :**

The function *limit_reps* places an internal restriction on the number of representations of **Op** that can be simultaneously maintained by GAMMA. By default this value is initially set to three for each operator, in anticipation of a probable need of **Op** in its default basis, its eigenbasis, and some other arbitrary basis. This should be adequate for most simulations so that *limit_reps* will not be needed.

Thus threre is then a possibility that three represenations of **Op** exist, but no more. If the user anticipates that his calculations will be switching between several bases and repeatedly using **Op** in them, the number of allowed representations should be increased. In such a case, **Op** mComputes the determinant of an operator.

Keep in mind that setting a higher limit does not allocate any more memory to an **Op** unless the operator is generated in different representations. For example

If, when *limit_reps* is invoked, the number of stored representatins of **Op** exceeds the newly set limiting value, representations will be deleted in order of low priority to high priority (see function *Op_priority*)

**Return Value:**

none

**Example(s):**

```
#include <gamma.h>
main()
  {
  gen_op Op1,Op2;                        // define two operators Op1 and Op2.
  limit_reps(Op1,100);                   // GAMMA may generate 100 reps. of Op1 if needed.
  limit_reps(Op2,1);                     // restricts GAMMA to maintain only 1 rep. of Op2.
  }
```

## 6.7.2 Op_priority

**Usage:**

#include <HSLib/GenOp.h>
void Op_priority (gen_op& Op, int i)

**Description :**

The function *Op_priority* places an internal priority on the current representation of **Op**. By default, **Op** is limited to three represenations.

**Return Value:**

none

**Example(s):**

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

## 6.7.3 set_limits

**Usage:**

```
#include <HSLib/GenOp.h>
void Op_priority (gen_op& Op, int i)
```

**Description :**

The function *Op_priority* places an internal priority on the current representation of **Op**. By default, **Op** is limited to three represenations.

**Return Value:**

none

**Example(s):**

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

# 6.8 Operator I/O Functions

## 6.8.1 <<

**Usage:**

#include <HSLib/GenOp.h>
friend ostream& operator << (ostream& ostr, const gen_op & Op)

**Description :**

If, when *limit_reps* is invoked, the number of stored representatins of **Op** exceeds the newly set limiting value, representations will be deleted in order of low priority to high priority (see function *Op_priority*)

**Return Value:**

none

**Example(s):**

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

## 6.9 Description

*Class OPERATOR* deals with each operator (**Op**) in terms of sub-units called **representations**. A representation of **Op** contains a matrix (*see Class MATRIX*) and a corresponding basis *(see Class BASIS)*. The matrix form of **Op** depends upon which basis the operator is expressed in. In fact, the operator matrix is useless without knowledge of its corresponding basis. *Class OPERATOR* will always associate an **Op** matrix with an **Op** basis and calls this pair a representation. (Keep in mind that a basis is itself a matrix, typically associated with the eigenbasis of some operator.) The figure below shows three likely representations of the same operator.

***Three Representations of Operator Op***

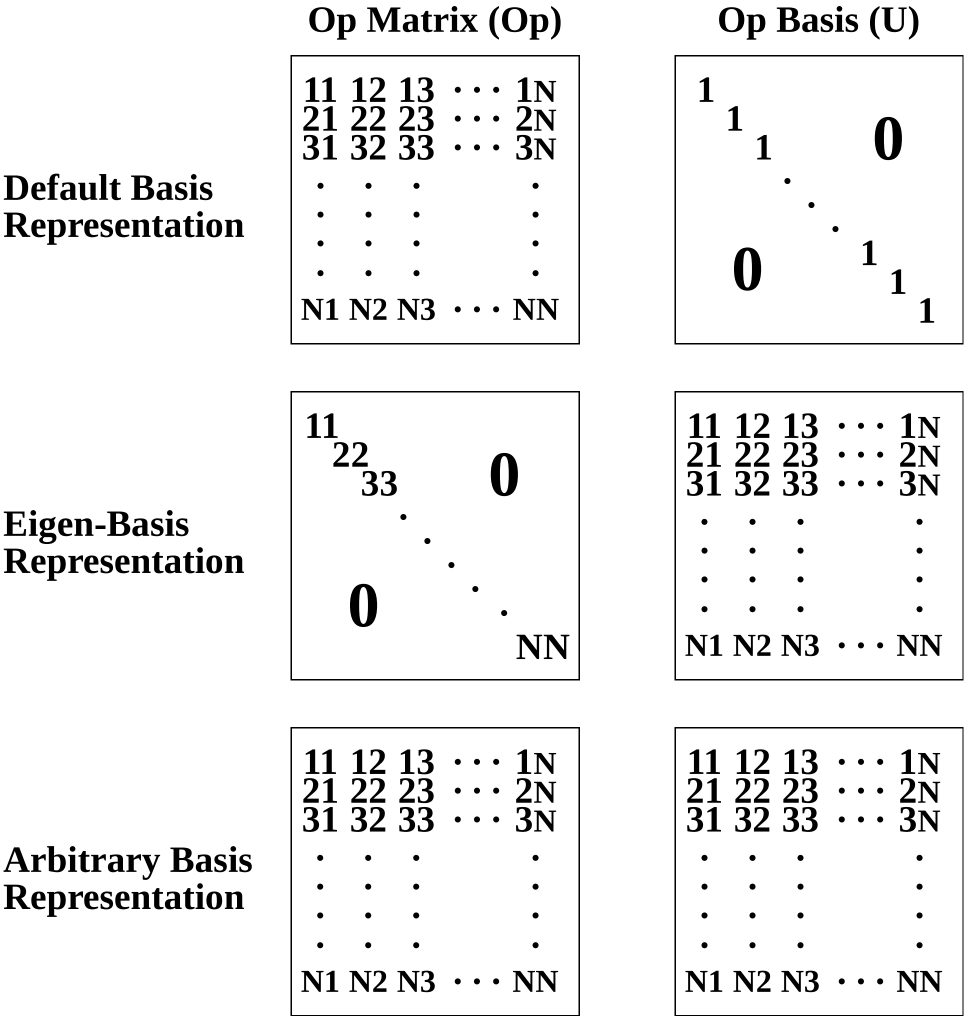


*Figure 4-3* **- Three representations of the same operator. In the default basis, the operator matrix may be completely filled but its basis is the identity matrix. When the operator is in its eigenbasis, the operator matix is diagonal and the basis matrix may be completely filled. In general, in any arbitrary basis, both the operator matrix and the basis matrix matrix will be filled.**

All representations of an operator contain the same information, the information is simply in a different format. Either due to computational simplicity or problem elucidation, specific representations are preferred over others and to switch between the different representations requires a similarity transformation. For example, the relationship between an operator, **Op**, in the default basis (superscript DB) and the same operator in an arbitrary basis (superscript AB) is given by the following equation.

$$(\boldsymbol{U^{AB}})^{\dagger}\boldsymbol{Op^{AB}U^{AB}} = \boldsymbol{Op^{DB}} \tag{9-2}$$

Here **U** is the transformation matrix or basis matrix (see *Class BASIS*) and **Op** the operator matrix. A representation is always a pairing of **Op** and **U**, *e.g.*

$$\{\boldsymbol{Op^{AB}}, \boldsymbol{U^{AB}}\} \equiv \boldsymbol{AB}\ \text{representation of}\ \boldsymbol{Op}. \tag{9-3}$$

In general, the relationship between a representation in two arbitrary bases (B1 and B2) is given by

$$\boldsymbol{U^{B}}^{2}[(\boldsymbol{U^{B}}^{1})^{\dagger}\boldsymbol{Op^{B}}^{1}\boldsymbol{U^{B}}^{1}](\boldsymbol{U^{B}}^{2})^{\dagger} = \boldsymbol{U^{B}}^{2}\boldsymbol{Op^{DB}}(\boldsymbol{U^{B}}^{2})^{\dagger} = \boldsymbol{Op^{B}}^{2} \tag{9-4}$$

Switching between the different representations of **Op** is thus seen to be a computationally expensive process as it involves at least two matrix multiplications. To avoid repeatedly switching back and forth between different representations *via* equation (7), GAMMA can maintain several representations of the same operator simultaneously. Once a representaion of **Op** is determined it is stored and then recalled (not calculated) upon subsequent usage. It should be emphasized that the different representations of **Op** can be transparent external to *Class OPERATOR*. That is, most calculations can make use of operators without knowledge of how GAMMA is managing operator representations. However, there can be a trade-off between memory use and computational efficiency. For more sophisticated and optimal use of GAMMA, it is advantageous to understand the internal workings of *Class OPERATOR.*

**Basic Structure** : We have already shown how each operator can have several representations simultaneously, as seen in the previous figure (Page xx). GAMMA has the ability to automatically generate and store different representations on demand, and does so with **dynamic memory allocation**. (How this is done will be demonstrated in the examples to follow shortly.) Three special represenations are always intenally tracked, the default basis representation DBR, the eigenbasis representation EBR, and the working basis representation WBR. DBR is the representation of **Op** whose transformation matrix is the identity matrix, **U** = **I**. EBR is the representation of **Op** whose matrix is diagonal. WBR is the current representation of **Op**, the last representation used in a calculation.

Consider the construction of an operator **Op** and it subsequent use with the following commands.

```
1. gen_op Op;
2. Op = Op1 + Op2;
```

The first line produces an operator with no allocation of array space whatsoever (it knows nothing about the dimension of the operator in any event!). This **Op** has no WBR, DBR, EBR, or any representations at all yet. It can be used in computations though, the second line will fill **Op** with one representation.

One can also construct an operator from a matrix, mx, the command would be

```
gen_op Op(mx);
```

This sets up potential array space for one representation of **Op**, its default basis representation DBR. Since the basis matrix is the identity matrix it is not stored as a full array. Since the matrix mx already uses array space **Op** shares the same memory as mx. If mx is modified, then will a new array be produced and **Op** have its own matrix space allocated for DBR. Since there is only one representation of **Op**, this is the WBR as well. If mx is a diagonal array, then this representation of **Op** will be the EBR also.

When using multiple operators, another memory saving scheme is utilized. Any operators having representations in a common basis will share the same basis matrix. The figure below shows three operators with representations in the same basis. Again, the basis matrix is labeled **U** (see *Class BASIS*) in keeping with past nomenclature and each representation is always an operator matrix - basis matrix pair.

***Three Operators Sharing the Same Basis***

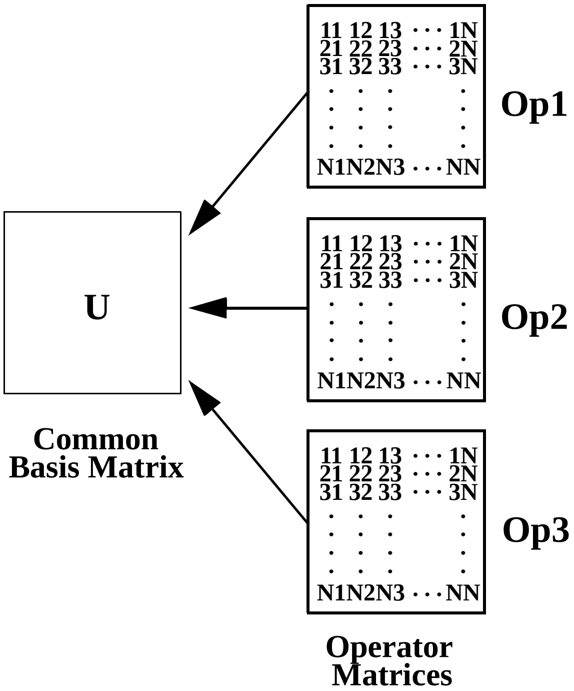


*Figure 4-4* **- Three operators which have representation in the same basis share the same basis matrix. The matrix cannot be altered and is used only to transform the operators into other representations. Computer memory use is conserved by maintaining the minimum number of basis matrices without loss of computation time.**

Operators with multiple representations will always share any and all common basis matrices. The diagram below depicts two operators, **A** and **B**, both having multiple representations and sharing some bases in common.

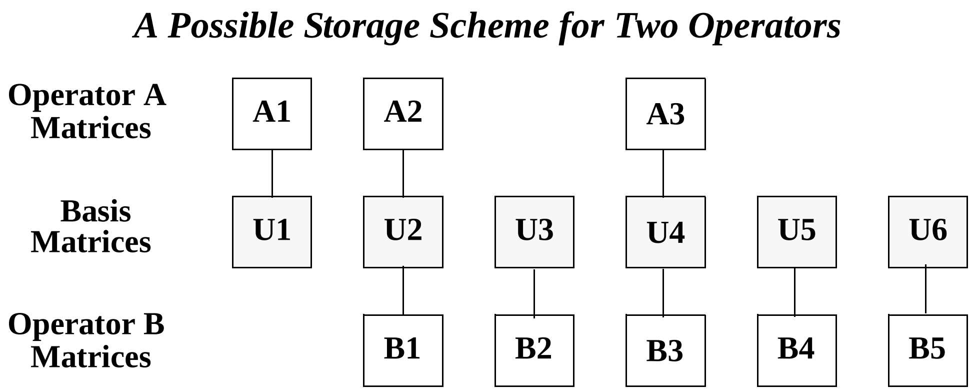


*Figure 4-5* **- The storage scheme used for multiple operators in several bases can become quite complicated. This example has operator A in two representations and operator B in 5 representations. The share three bases so the total number of basis matrices stored will be six.**

For calculations utilizing many operators and multipile bases, the matrix - basis relationships main

Note that the eigenbasis of one operator will not be the eigenbasis of another operator (unless they commute). There can also be overlap between DBR, EBR, and WBR of each operator.

The representation relationships maintained within *Class OPERATOR* can become very complicated. This is not usually the case as typically only a couple of bases are needed, but one might envision some extreme situations creating problems.

1. Too many representations of the same operator being intrinsically maintained by GAMMA.

2. A particular basis or set of bases being desired over other bases maintained by GAMMA.

Representation Limits : For each operator, no more than three representations will be internally maintained at any one time *unless the user specifically requests more*. This is to allow for facile computations in the default basis, the operator eigenbasis, and some arbitrary basis. Furthermore, no array space is allocated, for either the matrices or the bases, unless a representation is needed for a requested computation. If three representations are produced durIf several operators have representations in the same basis, only one basis (matrix ) is stored for all associated operators. These features of Class *OPERATOR* maximize program efficiency, both in terms of computation time and memory managment, while maintaining complete generality in operator definition and manipulation

Representation Priorities : Each representation of an operator has an assigned priority. Priorities tell *Class OPERATOR* how to preferentially maintain the represetations within the limits set by *max_reps*. For example, consider an operator **Op** which has its representation limit set to 2. Once two unique representations of **Op** have been needed, either specified or produced in the course of a calculation, the limit is reached. If a third representation is needed, GAMMA will overwrite the lowest priority representation currently maintained.

Priorities are assigned in three ways - by default, by computation, and by the user. GAMMA gives default values of 0 (lowest priority) to any representation it knows nothing about. It also will give a default value of 500 (temporary maximum) to the default basis represenation (DBR) and a value of (499) to the eigen-basis representation (EBR), *i.e.* it assumes these latter bases will most likely be repeatedly used in a computation. Priorities will also be transmitted during computation. If a representation is generated during a calculation involving other operators it adopts the highest priority of the represetations used to generate it. The user can always directly assign a priority to a specific operator representation. The value must be non-negative and may exceed the maximum internally set value of 500. This allows for the user demand GAMMA maintain certain bases preferentially.

*Class OPERATOR* has an internal knowledge of two special bases. The Default Basis is the basis to which all other operator representations are referenced to. All operators will share the same default basis even though they may not have a stored representation in that basis. The matrix for this basis is the identity matrix (see previous figure). The EigenBasis is the basis in which an operator matrix is diagonal. Two operators can have their eigenbase representations stored and not share the same basis matrix (*i.e.* not commute). If they commute they will share the same eigenbasis basis matrix. It may occure that the eigenbasis matrix is equivalent to the default basis matrix. Finally, the Working Basis is simply the last basis used for an operator. GAMMA assumes that further computations will need to be done in this basis and so keeps track of which operator representation was last used. All three bases corresponed to stored operator representations. The working basis will simply point to some other useful basis.

Since all functions available for the mathematical manipulation of operator quantities are defined within *Class OPERATOR*, internal checks are performed to insure that proper representations will always be used. For example, consider the addition of two operators to produce a third operator, **C = A** + **B**. **C** will be computed with representations of both operators in a common basis, that of the currently use representation of **A**. If a needed representation has not been previously computed, in this case **B** in the current basis of **A**, GAMMA will automatically compute and store it. After the addition, **C** will exist only one representation and shares a common basis with **A** and **B**. Any other previously existing representations of **C** will vanish.

As second example of the use of operator representations, consider the computation of an exponential operator, exp(**A**). GAMMA will automatically perform this calculation in the eigenbasis of **A** for maximum computational efficiency. If **A** is not in its eigenbasis representation when this funciton is invoked, GAMMA checks all currently maintained representations of **A**. If the eigenbasis representation already exists it is used, otherwise it is computed. The result is then transformed back into the basis of the original representation of **A** to produce exp(**A**) in the same basis of **A**.

A positive consequence of maintaining different representations of the same operator is that GAMMA aviods having to repeatedly switch representations through similarity transformations. This is computationally costly as it involves matrix multiplications. In GAMMA, a representation may be computed once, stored, and then recalled for subsequent use. A negetive consequence of of maintaining different representations of the same operator is that one could end up with many stored representations which will not be used in computations (other than the one for which they were initally created). This is an inefficient use of memory and can lead to serious problems in simulations involving many operators and many bases. Therefore, GAMMA has a built in strategy for effective memory management.

n internal and external and are computationaly costly. For each operator, no more than three representations (described above) will be stored at any one time *unless the user specifically requests more*. Furthermore, no array space is allocated, for either the matrices or the bases, unless a representation is needed for a requested computation. If several operators have representations in the same basis, only one basis (matrix ) is stored for all associated operators. These features of Class *OPERATOR* maximize program efficiency, both in terms of computation time and memory managment, while maintaining complete generality in operator definition and manipulation

# 7 Class Spin System

## 7.1 Overview

The class *Spin System* defines all the necessary physical attributes of a system of spins subjected to a stationary external magnetic field. Derived from the base class *Spin Sys, Spin System* has all the attributes of its base class (number of spins, spin angular momenta, and a defined Hilbert space) as well as information concerning spin environments, isotropic **chemical shifts** ($\nu$), isotropic **scalar couplings** (J), and an overall **spectrometer field strength** ($\Omega$).

## 7.2 Available Functions

### Spin System Algebraic

| | | |
|---|---|---|
| spin_system | - Constructor | page 157 |
| = | - Assignment, assign one spin system to another | page 166 |

### Basic Functions

| | | |
|---|---|---|
| shifts | - Set all chemical shifts to a specific value. | page 158 |
| shift | - Set or retrieve an individual chemical shift in Hertz. | page 158 |
| PPM | - Set or retrieve an individual chemical shift in PPM. | page 161 |
| Js | - Set all coupling constants to a specific value. | page 162 |
| J | - Set or retrieve an individual coupling constant. | page 162 |
| Omega | - Set or retrieve the spectrometer frequency[1]. | page 163 |

### Spin System Associated Functions

| | | |
|---|---|---|
| center | - Get a chemical shift center of the system. | page 165 |
| Nyquist | - Get an appropriate Nyquist frequency for the system. | page 165 |
| maxShift | - Get the maximum shift frequency of the system. | page 159 |
| minShift | - Get the minimum shift frequency of the system. | page 160 |
| medianShift | - Get an approximate Nyquist frequency for the system. | page 160 |
| lab_shift | - Get an individual chemical shift in the lab frame. | page 159 |
| offsetShifts | - Change all chemical shifts by the value indicated. | page 161 |

### Spin System I/O

| | | |
|---|---|---|
| = | - Assignment of a spin system to/from a p_set | page 166 |
| += | - Addition of a spin sys to a p_set | page 166 |
| write | - Write spin sys to disk file (as a parameter set). | page 169 |
| read | - Read spin sys from disk file (from a parameter set). | page 169 |
| print | - Send spin system to output stream. | page 168 |

[1] The function Omega also goes by the older name spectrometer_frequency. This latter name works fine but will no longer be maintained or documented.

| | | |
|---|---|---|
| << | - Send spin system to an output stream | page 168 |

Since class *spin_system* is derived from the base class *spin_sys* it maintains all the abilities of this class. Functions appropriate for spin system manipulation which are inherited from *spin_sys* are indicated below. For further details consult the chapter specific for that class.

### Inherited Functions (Base Class SpinSys)

| | | |
|---|---|---|
| spins | - Number of spins. | page 31 |
| spinpairs | - Number of spin pairs. | page 31 |
| HS | - Retrieve spin or spin system Hilbert space. | page 32 |
| isotope | - Set or retrieve spin isotope type. | page 33 |
| symbol | - Get spin isotope type as a string, e.g. 19F. | page 33 |
| qn | - Get spin angular momentum of a spin or the system. | page 34 |
| element | - Get spin element type as a string, e.g. Carbon. | page 34 |
| momentum | - Get spin angular momentum as a string. | page 35 |
| gamma | - Get gyromagnetic ration of a spin. | page 36 |
| qState | - Get the state vector of a particular quantum state. | page 36 |
| qStates | - Get a matrix with a description of all quantum states. | page 37 |
| qnState | - Get the quantum number of a state. | page 38 |
| qnStates | - Get the quantum number of all states. | page 39 |
| qnDist | - Get a statistic over the quantum numbers. | page 40 |
| CoherDist | - Get a statistic over the different coherences. | page 40 |
| homonuclear | - Test whether system is homonuclear. | page 41 |
| homonuclear | - Test whether system is heteronuclear. | page 41 |
| SetFlag | - Set a spin flag to true or false. | page 45 |
| SetFlags | - Set selected spin flags to true or false. | page 45 |
| GetFlag | - Get a spin flag. | page 46 |
| GetFlags | - Get spin flags. | page 46 |
| name | - Set or retrieve spin system name. | page 47 |

Keep in mind that functions which may be applied to spin systems are inherited from the base class *spin_sys*. As functions are added to class *spin_sys* they automatically are added (inherited) here. Thus there may be other functions in *spin_sys* which are useful for certain applications yet not documented in this chapter. Check the documentation for *spin_sys* for additional functions. Furthermore, when a certain function is desired for the manipulation of a spin system the user should ask whether the function is specific for a spin system or more generally applicable for a spin_sys. If the later is true the new function(s) should be added to the base class rather than here. The inheritance will of course immediately make it applicable here also.

# 7.3 Algebraic Functions

## 7.3.1 spin_system

**Usage:**

```
#include <HSLib/SpinSystem.h>
void spin_system::spin_system(int nspin=0)
void spin_system::spin_system(spin_system &sys)
```

**Description:**

The function ***spin_system*** is used to create a new spin system.

1. spin_system(int nspins=0) - Called with an integer, the function will create a spin system of the size indicated. By default, all spins are set to be protons. If the number of spins is not supplied as an argument the function creates an "empty" NULL spin system. A spin system constructed with zero spins will not have full spin system capabilities until either the number of spins has been assigned or the system itself has been set equal to another spin system.
2. spin_system(spin_system &sys) - Called with another parameter set, the function will make a new spin system which is a copy of the given spin system.

**Return Value:**

***spin_system*** returns no parameters. It is used strictly to create a spin system object.

**Examples:**

```
#include <gamma.h>
main()
  {
  spin_system A;                    // define all NULL spin system called A.
  spin_system A2BX(4);              // define spin system A2BX containing four spins.
  spin_system Four(A2BX);           // define spin system Four identical to current A2BX.
  A = Four;                         // Valid way to set system A equal to current Four.
  spin_system xyz(5);               // define a second spin system xyz with five spins.
  //xyz.spin_system(5);             // alternative equivalent statement as previous line.
  }
```

**See Also: = (the assignment operator)**

# 7.4 Basic Functions

## 7.4.1 shifts

**Usage:**

```
#include <HSLib/SpinSystem.h>
void spin_system::shifts(double) const
```

**Description:**

Function ***shifts*** sets **all** spin chemical shifts to the value input. Input units are assumed to be Hertz. This is useful when it is necessary to zero all values.

**Return Value: None.**

**Examples:**

```
#include <gamma.h>
main()
  {
  spin_system sosi(3);                    // define a three spin system called sosi.
  sosi.shifts(1027.258);                  // set shifts of all spins to 1027.258 Hertz.
  }
```

**See Also: shift, PPM**

## 7.4.2 shift

**Usage:**

```
#include <HSLib/SpinSystem.h>
double spin_system::shift(int) const
void spin_system::shift(int, double)
```

**Description:**

Function ***shift*** is used to set/retrieve chemical shift values.

1. shift (int) - Called only with an integer, the function returns a double which is the value of the chemical shift of spin int in Hertz. For an N spin system the range of int = [0, N-1].
2. shift (int, double) - Called with both and integer and a double, the function will store the value of double as the chemical shift of spin int in Hertz. For an N spin system the range of int = [0, N-1].

**Return Value:**

None in setting the shift, double when retrieving a shift.

**Examples:**

```
#include <gamma.h>
main()
  {
  double v0, v4;
  spin_system xyz(5);                     // define spin system xyz containing five spins.
  xyz.shift(0, -123.67);                  // set chemical shift of xyz spin 0 to -123.67 Hz.
  v0 = xyz.shift(0);                    // retrieve chemical shift of xyz spin 0 (-123.67 Hz).
  v4 = xyz.shift(4);                    // retrieve chemical shift of xyz spin 4 (0 Hz, default).
  }
```

**See Also: shifts, PPM**

## 7.4.3 lab_shift

**Usage:**

```
#include <HSLib/SpinSystem.h>
double spin_system::lab_shift(int) const
```

**Description:**

Function ***lab_shift*** is used to retrieve chemical shift values as seen from the laboratory frame. The function takes an integer argument for the index of the spin for which the shift is desired. The output value will be in Hertz, typically on the scale of the spectrometer frequency ($10^8$).

**Return Value:**

A double.

**Examples:**

```
#include <gamma.h>
main()
  {
  spin_system sys;                    // Define spin system sys.
  sys.read("filename");               // Read in the system parameters from an external file.
  cout << sys.lab_shift(0);           // Output the laboratory frame shift of the 1st spin.
  }
```

**See Also: shift, PPM**

## 7.4.4 maxShift

**Usage:**

```
#include <HSLib/SpinSystem.h>
double spin_system::maxShift() const
```

**Description:**

Function ***maxShift*** is used the retrieve the maximum chemical shift value (Hertz) of all spins in the system.

**Return Value:**

A double precision number.

**Examples:**

```
#include <gamma.h>
main()
  {
  spin_system sys;                    // define an empty spin system.
  sys1.read("filename");              // read sys in from external file called filename
  cout << "\nMax: " << sys.maxShift;  // Output the largest frequency in the system
  }
```

**See Also: minShift, shift, PPM**

### 7.4.5 minShift

**Usage:**

```
#include <HSLib/SpinSystem.h>
double spin_system::minShift ()
```

**Description:**

The function ***minShift*** is used the retrieve the minimum chemical shift value of all spins in the system. The value is returned in Hertz.

**Return Value:**

A double precision number.

**Examples:**

```
#include <gamma.h>
main()
  {
  spin_system sys;                          // define an empty spin system.
  sys1.read(“filename”);                    // read sys in from external file called filename
  double min = sys.minShift();              // retrieve the smallest frequency in the system
  }
```

**See Also: maxShift, shift, PPM**

### 7.4.6 medianShift

**Usage:**

```
#include <HSLib/SpinSystem.h>
double spin_system::medianShift ()
```

**Description:**

The function ***medianShift*** is used the retrieve median chemical shift value of the system. The value is returned in Hertz. This is useful to find the center frequency of a spectrum

**Return Value:**

A double precision number.

**Examples:**

```
#include <gamma.h>
main()
  {
  spin_system sys;                          // define an empty spin system.
  sys1.read(“filename”);                    // read sys in from external file called filename
  double midpt = sys.medianShift();         // retrieve the median frequency in the system
  }
```

**See Also: maxShift, minShift, shift, PPM**

## 7.4.7 offsetShifts

**Usage:**

```
#include <HSLib/SpinSystem.h>
void spin_system::offsetShifts (double)
```

**Description:**

The function ***offsetShifts*** changes all chemical shift values in the spin system by the value indicated.

**Return Value:**

None.

**Examples:**

```
#include <gamma.h>
main()
  {
  spin_system sys;                         // define an empty spin system.
  sys.read(“filename”);                    // read sys in from external file called filename
  double midpt = sys.medianShift();        // retrieve the median frequency in the system
  sys.offsetShifts(midpt);                 // offset system to be frequency centered
  }
```

**See Also: maxShift, minShift, medianShift, shift, PPM**

## 7.4.8 PPM

**Usage:**

```
#include <HSLib/SpinSystem.h>
double PPM (int)
void PPM (int, double)
```

**Description:**

The function ***PPM*** is used for both to set or obtain the chemical shift value of a spin in PPM. A spectrometer frequency must be specified prior to use of this function (see *spectrometer_frequency*). The following formats are defined within the class spin system.

1. 1. PPM (int) - Called only with an integer, the function returns a double which is the value of the chemical shift of the spin in PPM. For an N spin system the range of int = [0, N-1].
2. 2. PPM (int, double) - Called with both and integer and a double, the function will assign the value input as the chemical shift of the spin indicated in PPM. For an N spin system the range of int = [0, N-1].

PPM values are computed with the formula

$$\delta_{PPM}(i) = (\delta_{Hz}(i)) / \left[\frac{\gamma(i)}{\gamma(H)} \times \Omega_{MHz}\right] \quad ,$$

where $\delta$’s are the chemical shifts, $\gamma$’s the gyromagnetic ratios and $\Omega$ the proton based spectrometer frequency.

**Return Value:**

None in setting the shift, double when retrieving a shift.

**Examples:**

```
  #include <gamma.h>
  main()
    {
    double v1, v2, v3;
    spin_system xyz(5);                 // define spin system xyz containing five spins.
    xyz.shift(1, -100.0);               // set chemical shift of xyz spin 1 to -100.0 Hz.
    xyz.shift(2, 200.0);                // set chemical shift of xyz spin 2 to 200.0 Hz.
    // xyz.PPM(3, 5.3);                  - Not allowed, no spectrometer frequency.
    xyz.Omega(100.0);                   // set spectrometer frequency to 100 MHz.
    xyz.PPM(3, 5.0);                    // set chemical shift of xyz spin 3 to 5 PPM.
    xyz.PPM(2, 4.5);                    // re-set chemical shift of xyz spin 2 to 4.5 PPM.
    v1 = xyz.PPM(1);                    // retrieve chemical shift in PPM of xyz spin 1 (-1).
    v2 = xyz.PPM(2);                    // retrieve chemical shift in PPM of xyz spin 2 (4.5).
    v3 = xyz.PPM(3);                    // retrieve chemical shift in PPM of xyz spin 3 (5.0).
    }
```

**See Also: shift, Omega**

## 7.4.9 Js

**Usage:**

```
#include <HSLib/SpinSystem.h>
void spin_system::Js (double)
```

**Description:**

The function ***Js*** is used to set all the scalar couplings in a spin system to the same value. Input units are assumed to be Hertz. This is useful when it is necessary to zero all values.

**Return Value:**

None.

**Examples:**

```
  #include <gamma.h>
  main()
    {
    spin_system sys1;                   // Define a spin system.
    sys1.read(“filename”);              // read sys1 in from external file called filename
    sys1.Js(0.0);                       // set all J’s to zero
    }
```

**See Also: shifts**

## 7.4.10 J

**Usage:**

```
#include <HSLib/SpinSystem.h>
double spin_system::J (int, int)
void spin_system::J(int, int, double)
```

### Description:

Function ***J*** is used to set/obtain scalar coupling constants between spins.

1. J(int1, int2) - returns the scalar coupling constant in Hertz between spin int1 and spin int2.
2. J(int1, int2, double) - stores the value of double as the scalar coupling constant in Hertz between spin int1 and spin int2.

### Return Value:

None when setting the constant, double when retrieving the constant.

### Examples:

```
#include <gamma.h>
main()
  {
  double J12, J13;
  spin_system x(5);                    // define spin system x containing five spins.
  x.J(1, 2, -3.7);                     // set J(1,2) of spin system x to -3.7 Hz.
  x.J(1, 3, 6.35);                     // set J(1,3) of spin system x to 6.35Hz
  J12 = x.J(1, 2);                     // retrieve J(1,2) of spin system x (-3.7Hz).
  J13 = x.J(1, 3);                     // retrieve J(1,3) of spin system x (6.35Hz).
  }
```

**See Also: shift, PPM**

## 7.4.11 Omega[1]

### Usage:

```
double spin_system::Omega ( )
void spin_system::Omega(double)
```

### Description:

The function ***Omega*** is used to set and retrieve the spectrometer field strength. The field strength is specified in terms of the base **proton** frequency, typically given in MHz. The field strength is used to determine chemical shifts in PPM and to determine transition frequencies on an absolute (MHz) scale for some computations involving relaxation. The value of Omega need not be assigned if unnecessary for the specific calculation.

The following formats are defined within the class spin system:

1. Omega( ) - returns the proton based spectrometer frequency in MHz.
2. Omega(double) - stores the value of double as the proton based spectrometer frequency in MHz.

### Return Value:

Double, proton based spectrometer field strength in MHz.

### Example(s):

```
#include <gamma.h>
main()
{
  ABX.spin_system(3);                  // define spin system ABX containing three spins.
```

[1] The function Omega also goes by the older name spectrometer_frequency. This latter name works fine but will no longer be maintained or documented.

```
ABX.spectrometer_frequency(220);         // set 1H based spectrometer frequency to 220 MHz.
cout << ABX.spectrometer_frequency();   // print 1H based spectrometer frequency (220).
ABX.spectrometer_frequency(220, ‘G’);   // set spectrometer frequency to 220 GHz!
cout << ABX.spectrometer_frequency();   // print spectrometer frequency (220000).
}
```

**See Also: isotope, shift.**

# 7.5 Associated Functions

## 7.5.1 center

**Usage:**

```
#include <HSLib/SpinSystem.h>
double spin_system::center(int spin=0)
```

**Description:**

The function ***center*** is used to obtain the frequency of the spin system based on the chemical shifts of all spins of the same isotope type as the input ***spin***. For homonucelar system a value for ***spin*** need not be given. The returned value is always in Hertz

**Return Value:**

A double.

**Examples:**

```
#include <gamma.h>
main()
  {
  spin_system sys;                        // define spin system sys.
  sys.read(“file.sys”);                   // read spin system in from file “file.sys”.
  double cent = sys.center();             // set variable cent to the frequency center (spin 0).
  }
```

**See Also: maxShift, minShift, Nyquist**

## 7.5.2 Nyquist

**Usage:**

```
#include <HSLib/SpinSystem.h>
double spin_system::Nyquist (int spin=0, double fact=1.1, double lwhh=0.0)
```

**Description:**

The function ***Nyquist*** is used to obtain an approximate Nyquist frequency for the spin system over all spins of the same isotope type as the input ***spin***. The returned value estimates the largest absolute transition frequency based on shifts and scalar couplings. The value of ***fact*** is used to shrink or expand the output value, the default value adds 10% to the value estimated for the largest transition. When a broad line is anticipated, an approximate linewidth at half-height can be input as ***lwhh*** (in Hertz) and will be taken into account. The returned value is in Hertz.

**Return Value:**

A double.

**Example:**

```
#include <gamma.h>
main()
  {
  spin_system sys;                        // define spin system sys.
  sys.read(“file.sys”);                   // read spin system in from file “file.sys”.
  double Nyqf = sys.Nyquist(0);           // set Nyqf to the approx. Nyquist frequency over
                                          // spins of the same type as the first spin.
  }
```

**See Also: maxShift, minShift**

### 7.5.3 =

**Usage:**

```
#include <HSLib/SpinSystem.h>
void spin_system::operator= (const spin_system&)
void operator= (p_set&, const spin_system&)
void operator= (p_set&)
```

**Description:**

The ***operator*** **=** is used here to equate one spin system to another, equate a spin system to a parameter set, or equate a parameter set to a spin system.

1. spin_system::operator= (const spin_system&) - This usage equates one spin system to another. Any information currently in the spin system to which the assignment is made will be destroyed.
2. operator= (p_set&, const spin_system&) - Assigns a spin system to a parameter set. Any parameters in the p_set prior to assignment are deleted. The resulting p_set will contain only the spin system parameters.
3. operator= (p_set&, const spin_system&) - Assigns a parameter set to a spin system. Any information currently in the spin system to which the assignment is made will be destroyed. Only parameters pertaining to the spin system will be taken from the parameter set (See section 7.8 on page 171 for valid spin system parameters.). Parameters not found in the parameter set will be set to default values.

**Return Value:**

None.

**Example(s):**

```
#include <gamma.h>
main()
  {
  spin_system sys1, sys2;                 // define two spin systems, sys1 and sys2.
  sys1.read(“file.sys”);                  // read spin system sys1 in from file “file.sys”.
  sys2 = sys1;                            // set spin system sys2 equal to spin system sys1
  p_set pset;                             // define a parameter set
  pset = sys2;                            // parameter set pset now contains all sys2 info.
  sys1 = pset;                            // sys1 now set to system defined in pset
  }
```

**See Also:** +=

### 7.5.4 +=

**Usage:**

```
#include <HSLib/SpinSystem.h>
void operator+= (p_set&, const spin_system&)
```

### Description:

The ***operator*** **+=** is used to add all spin system parameters to an existing parameter set.

### Return Value:

None. The parameter set is modified by the additon of spin system parameters.

### Example:

```
#include <gamma.h>
main()
  {
  spin_system sys;                      // define spin system sys.
  sys.read(“file.sys”);                 // read spin system in from file “file.sys”.
  p_set pset;                           // define a parameter set
  pset += sys;                          // add spin system parameters to the parameter set
  }
```

**See Also: =**

# 7.6 I/O Functions

## 7.6.1 <<

**Usage:**

```
#include <HSLib/SpinSystem.h>
ostream& operator<< (ostream&, spin_system&)
```

**Description:**

The ***operator*** << is used for standard output of a spin system. This includes chemical shifts, coupling constants, isotope labels, spin quantum numbers, etc.

**Return Value:**

None.

**Example(s):**

```
#include <gamma.h>
main()
  {
  CH3.spin_system(3);                    // define spin system CH3 containing three spins.
  cout << CH3;                           // write the spin system CH3 to standard output.
  }
```

**See Also: write, read, print**

## 7.6.2 print

**Usage:**

```
#include <HSLib/SpinSystem.h>
void print( )
```

**Description:**

The function ***print*** can be used to send all the current information stored in a spin system to an output stream. This includes chemical shifts, coupling constants, isotope labels, spin quantum numbers, etc.

**Return Value:**

None.

**Example(s):**

```
#include <gamma.h>
main()
  {
  CH3.spin_system(3);                    // define spin system CH3 containing three spins.
  CH3.print( );                          // print all information contained in spin system CH3.
  }
```

**See Also:**

## 7.6.3 write

**Usage:**

```
#include <HSLib/SpinSystem.h>
void spin_system::write(const string& filename);
```

**Description:**

The function ***write*** enables users to write the spin system out to an ASCII file which is readable by both the user and by the analogous spin system function *read*. Spin system parameters are written in the standard parameter set format (see the Section Spin System Parameter Files later in this Chapter).

**Return Value:**

None.

**Example:**

```
#include <gamma.h>
main()
  {
  spin_system sys(3);                    // define spin system sys containing three spins.
  sys.write("test.sys");                 // write out spin system test to file test.sys.
  }
```

**See Also: read**

## 7.6.4 read

**Usage:**

```
#include <HSLib/SpinSystem.h>
void spin_system::read (const string&filename);
```

**Description:**

The function ***read*** is enables users to read in a spin system from an external file. The file is assumed to be written in the standard parameter set format (see the Section Spin System Parameter Files later in this Chapter) utilizing the standard parameter names associated with a spin system. Typically, the file being read was generated from a previous simulation through the use of the analogous spin system function *write*. Alternatively, one can use an editor to construct a parameter set file for the spin system.

**Return Value:**

None.

**Example:**

```
#include <gamma.h>
  main()
  {
  spin_system sys;                       // define an empty spin system.
  sys.read("test.sys");                  // read in the spin system from the file test.sys.
  }
```

**See Also: write**

## 7.7 Description

The fundamental quantities which describe a collection of spins (**number of spins**, **spin angular momenta**) are intrinsic to the class *spin_sys*. In many applications, such as in the treatment of magnetic resonance problems, one also desires information concerning spin environment. In particular, for NMR constants such as isotropic **chemical shifts** ($\nu$), isotropic **scalar couplings** (J), and **spectrometer field strength** ($\Omega$) are very important. Such information is contained in class *spin_system*. Class *spin_system* is derived from class *spin_sys* had has all of its properties and functional capabilites. Additionally, *spin_system* contains these environment constants and functions to easily manipulate them.

The chemical shift in Hertz of any spin in the system can be both specified and accessed by the function *shift*. In analogy, the function *PPM* allows for chemical shift specification in PPM units, assuming the spectrometer frequency is present. Scalar couplings are set and obtained (in Hertz) through the function *J*. The spectrometer frequency may be accessed with the function *Omega*..

Several functions are provided for facile I/O of spin systems. The function *read* is provided for spin system input from an external file. The analogous output function *write* sends the spins system to an ASCII output file which is readable by the user as well as the function *read*. The standard C++ output function << is also included and the function *print* which will send spin system information to any output stream.

***Spin System Internal Structure***

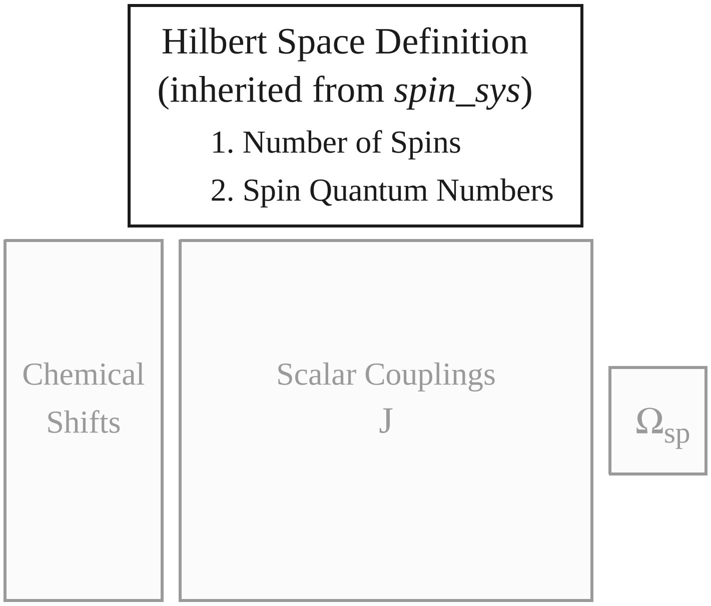


***Figure 4-6*** **Spin systems are derived from the base class SpinSys (spin_sys). This sets up an associated spin Hilbert space. In addition, these systems contains a field strength (proton Larmor frequency), a set of isotropic scalar/hyperfine and hyperfine coupings, and a set of isotropic shifts/g-factors.**

## 7.8 Parameter Files

This section describes how an ASCII file may be constructed that is self readable by a spin system. The file can be created with an editor of the users choosing and is read with the spin system member function “read”. Examples of such spin system files are given in a later section of this Chapter.

A spin system input file is scanned for the specific parameters which specify the spin system[1]: number of spins, isotope types, chemical shifts, coupling constants, etc. Parameters important to the spin system are recognized by certain keywords, as shown in the following table.

**Table 2: Spin System Parameters**

| Parameter Keyword | Assumed Units | Examples<br>Parameter (Type) : Value - Statement |
|---|---|---|
| SysName | none | SysName (2) : Glycine - Spin System Name |
| NSpins | none | Nspins (0) : 5 - Number of Spins |
| Iso(i) | none | Iso(0) (2) : 19F - Spin Isotope Type<br>Iso(1) (2) : 3H - Spin Isotope Type<br>Iso(3) (2) : 13C - Spin Isotope Type |
| Omega | MHz | Omega (1) : 270.0 - Spectrometer Frequency |
| v | Hz | v(0) (2) : 2.37 - Chemical Shift (Hz) |
| PPM | PPM | PPM(0) (2) : 113.2 - Chemical Shift (PPM) |
| J | Hz | J(0,1) (2) : 2.37 - Coupling Constant (Hz)<br>J(0,3) (2) : 12.1 - Coupling Constant (Hz)<br>J(2,3) (2) : 8.06 - Coupling Constant (Hz) |

The format of each parameter is quite simple. At the beginning of a line the keyword is written followed by some blanks and then an integer in parentheses. The integer corresponds to the type of parameter value: 0 = integer, 1 = floating point, or 2 = string. Following the parenthesis should be a colon to indicate the parameter value follows. The parameter value is then written followed by some blanks then a hypen followed by an optional comment.

There is one major restriction; keywords and string parameters cannot contain blanks. For example, v (0) is unknown, v(0) is. The string value 19 F is unknown, 19F is fine. A system name of Glutamic Acid would be seen as Glutamic. Glutamic_Acid would be more suitable.

To read the file, see the documentation for function read. There is also an example program read-

[1] Long parameter files, of which spin system parameters are only a small percentage, may be maintained by the user for various applications. When a spin system is read it will ignore parameters it does not recognize and continue searching for parameters it knows.

system.cc provide at the end of this Chapter which should indicate how the file is read. Each of the possible spin system input parameters is now described in more detail.

### Spin system name: SysName (inherited from class spin_sys)

A spin system name may be entered. It has no mathematical function in GAMMA but comes in handy when quickly scanning the input file or in tagging some output file with the spin system name. This parameter is optional. Parameter type 2 indicates a string parameter.

### Number of spins: Nspins (inherited from class spin_sys

It is essential that the number of spins be specified. A simple integer is input here. This is the first thing that will be read from the file so keep it somewhere near the top. Parameter type 0 indicates an integer parameter.

### Isotope Types: Iso(i) (inherited from class spin_sys)

Spin isotopes are specified with this parameter. These are optional, spins for which there is no isotope specified will be set to protons. A complete lookup table of isotope information is internal to GAMMA and scanned upon spin system construction depending upon the isotope types. Most actual isotopes are included in GAMMA, for a full list see the documentaion for class spin_sys. Isotopes are specified by the atomic number immediately followed (no blanks) by the atomic symbol. Note that the first spin is spin zero and the last spin is spin N-1 in a spin system with N spins. The values need not be in any order in the file nor even following each other. Parameter type 2 indicates a string parameter.

### Spectrometer Frequency: Omega

A spectrometer frequency should be specified. GAMMA assumes the input value is in MHz and associated with the Larmor precessional frequency of protons. If no frequency is present the value will be set to 500 MHz automatically. Parameter type 1 indicates a floating point number as the parameter value.

### Chemical Shifts: v(i), PPM(i)

Isotropic chemical shifts should be input for each spin which have no shielding tensors. If a shielding tensor is present in the file it will be preferentially read and the value input here skipped. Isotropic shifts can be input either in Hertz (with parameter v) or in PPM (with parameter PPM) the latter only accepted if the spectrometer frequency has been specified. All shift information is internally maintained in GAMMA in Hertz Note that the first spin is spin zero and the last spin is spin N-1 in a spin system with N spins. The values need not be in any order in the file nor even following each other. When these are written by GAMMA to a similar file they will be output as correlation times in nanoseconds. Parameter type 2 indicates a string parameter.

### Coupling Constants: J(i,j)

Isotropic scalar coupling constants can be input for each spin pair which have no scalar coupling tensor. If a scalar coupling tensor is present this parameter is ignored. J is input in Hertz. Note that the first spin is spin zero and the last spin is spin N-1 in a spin system with N spins. The values need not be in any order in the file nor even following each other. Parameter type 1 indicates a floating point parameter value.

# 7.9 Example Parameter Files

Galactose

```
SysName (2) : galact          - Name of the Spin System (galactose)
NSpins  (0) : 8               - Number of Spins in the System
Iso(0)  (2) : 1H              - Spin Isotope Type
Iso(1)  (2) : 1H              - Spin Isotope Type
Iso(2)  (2) : 1H              - Spin Isotope Type
Iso(3)  (2) : 1H              - Spin Isotope Type
Iso(4)  (2) : 1H              - Spin Isotope Type
Iso(5)  (2) : 1H              - Spin Isotope Type
Iso(6)  (2) : 1H              - Spin Isotope Type
Iso(7)  (2) : 1H              - Spin Isotope Type
PPM(0)  (1) : 4.27            - Chemical Shifts in PPM
PPM(1)  (1) : 3.47            - Chemical Shifts in PPM
PPM(2)  (1) : 3.6             - Chemical Shifts in PPM
PPM(3)  (1) : 3.89            - Chemical Shifts in PPM
PPM(4)  (1) : 3.63            - Chemical Shifts in PPM
PPM(5)  (1) : 3.76            - Chemical Shifts in PPM
PPM(6)  (1) : 3.71            - Chemical Shifts in PPM
PPM(7)  (1) : 3.54            - Chemical Shifts in PPM
J(0,1)  (1) : 7.8             - Coupling Constants in Hz
J(1,2)  (1) : 8.2             - Coupling Constants in Hz
J(2,3)  (1) : 3.7             - Coupling Constants in Hz
J(3,4)  (1) : 4.0             - Coupling Constants in Hz
J(4,5)  (1) : 10.5            - Coupling Constants in Hz
J(5,6)  (1) : 10.5            - Coupling Constants in Hz
J(6,7)  (1) : 15              - Coupling Constants in Hz
Omega   (1) : 400             - Spectrometer Frequency in MHz (1H based)
```

# 7.10 Example Source Codes

READSYSTEM - Program for reading a spin system from a file

```
/***************************************************************
**                                                            **
** This GAMMA program runs interactively. It will ask the **
** user for a filename which contains the spin system. Then**
** it will attempt to read the spin_system from the in-       **
** indicated disk file. If successful it will send the        **
** spin system parameters to standard output (the screen).   **
**                                                            **
***************************************************************/

#include <gamma.h>

main (int argc, char*argv[])
{
 spin_system sys;
 string filename;
 query_parameter(argc, argv, 1,
 “Spin system filename? “, filename);
 sys.read(filename);
 cout << sys;
}
```

# 8 Hamiltonians

## 8.1 Overview

The functions herein provide the users of GAMMA with easy access to some of the commonly used static Hamiltonians in NMR. There are two basis types of Hamiltonians available in GAMMA. The first are the static Hamiltonians due to the isotropic, rotationally invariant parts of common interactions: Zeeman, chemical shielding, scalar coupling. The second type of Hamiltonian returned is formed from blending spatial and spin tensors. These are not rotationally invariant and not "static" if the system is moving. Access functions are provided to rotate these Hamiltonians into any coordinate system of choice. The code is implemented for spin systems containing spins of any I quantum number.

## 8.2 Available Hamiltonian Functions

| | | |
|---|---|---|
| Ho | - High-resolution isotropic NMR Hamiltonian | page 177 |
| How | - High-resolution isotropic NMR Hamiltonian | page 177 |
| Ho_lab | - High-resolution isotropic NMR Hamiltonian, Laboratory Frame | page 178 |
| Hcs | - Isotropic chemical shielding Hamiltonian | page 178 |
| Hcs_lab | - Isotropic chemical shielding Hamiltonian, Laboratory Frame | page 178 |
| HJ | - Isotropic scalar coupling Hamiltonian | page 181 |
| HJw | - Isotropic scalar coupling Hamiltonian, weak coupling | page 182 |
| HJwh | - Isotropic scalar coupling Hamiltonian, weak heteronuclear | page 182 |
| Hz | - Isotropic scalar coupling Hamiltonian, weak coupling | page 184 |
| H1 | - Applied rf-field Hamiltonian | page 186 |
| Heff | - Effective Hamiltonian under an Applied rf-field | page 186 |

## 8.3 Covered Hamiltonian Theory

| | | |
|---|---|---|
| $\boldsymbol{H}^{Z}$ | - The Zeeman Hamiltonian | page 191 |
| $\boldsymbol{H}^{CS}$ | - The Chemical Shielding Hamiltonian | page 202 |
| $\boldsymbol{H}^{D}$ | - The Dipolar Hamiltonian | page 220 |
| $\boldsymbol{H}^{J}$ | - The Scalar Coupling Hamiltonian | page 228 |
| $\boldsymbol{H}^{Q}$ | - The Quadrupole Hamiltonian | page 238 |
| $H^{RDM}$ | - The Random Field Hamiltonian | page 249 |

$H_{cs}$ - The Chemical Shift Hamiltonian page 255
$H_0$ - The High Resolution NMR Hamiltonian page 256
$H$ - The RF Field Hamiltonian page 257

## 8.4 Hamiltonian Figures

Magnetization Response to the Shift Hamiltonian page 180
Magnetization Response to Static Magnetic Field page 185
Heff Eigenvalues versus B1 Field Strength page 188
Rank 1 Irreducible Spherical Spin Tensor Components page 192
Rank 1 Irreducible Spherical Spin Tensor Component Matrix Representations page 192
Rank 1 Zeeman Irreducible Spherical Spatial Tensor Components page 193
Field Aligned Rank 1 Zeeman Irreducible Spherical Spatial Tensor Components page 193
Zeeman Rank 2 Irreducible Spherical Spin-Space Tensor Components page 195
Rank 2 Zeeman Spherical Spin-Space Tensor Component Matrix Representations page 196
Aligned Rank 2 Zeeman Irreducible Spherical Spin-Space Tensor Components page 196
Aligned Rank 2 Zeeman Spin-Space Tensor Component Matrix Representations page 197
Zeeman Rank 2 Irreducible Spherical Spatial Tensor Components (PAS) page 199
Zeeman Rank 2 Treatment Hamiltonian Summary page 205
Rank 1 Irreducible Spherical Spin Tensor Components page 205
Zeeman Hamiltonian Summary page 205
Rank 1 Chemical Shielding Irreducible Spherical Spatial Tensor Components page 205
The Rank 1 Chemical Shielding Hamiltonian Summary page 205
Rank 1 Chemical Shielding Irreducible Spherical Spatial Tensor Components page 205
Rank 1 Chemical Shielding Irreducible Spherical Spatial Tensor Components page 205
Quadrupolar Irreducible Spherical Spin Tensor Components page 243
Quadrupolar Spherical Spin Tensor Components Matrix Representations page 243
Quadrupolar Irreducible Spherical Spatial Tensor Components page 248
The Quadrupolar Hamiltonian Summary page 248

## 8.5 Hamiltonian Example Programs

The Chemical Shift Hamiltonian page 261
The Zeeman Hamiltonian page 264
Effective Hamiltonian page 266
Dipolar Hamiltonian Spin Tensors page 267
Quadrupolar Hamiltonian Spin Tensors page 269

# 8.6 Routines

## 8.6.1 Ho

**Usage:**

```
#include <nmr_ham.h>
gen_op Ho(spin_system&);
```

**Description:**

The function ***Ho*** returns a general operator containing the standard isotropic liquid NMR Hamiltonian belonging to the spin system specified by the argument. The operator returned has frequency units and is computed by the formula

$$\boldsymbol{H}_o = \boldsymbol{H}_{cs} + \boldsymbol{H}_{Jwh}, \tag{9-5}$$

where $\boldsymbol{H}_{cs}$ is the isotropic chemical shift Hamiltonian in the (multiply) rotating frame and $\boldsymbol{H}_{Jwh}$ isotropic scalar coupling Hamiltonian which implicitly sets heteronuclear couplings as being weak.

**Return Value:**

Isotropic liquid NMR Hamiltonian as a general operator in frequency units.

**Example(s):**

```
#include <gamma.h>
spin_system(ab);                    //
gen_op Ham;                         //
Ham = Ho(ab);                       //
```

**See Also:**

## 8.6.2 How

**Usage:**

```
#include <nmr_ham.h>
gen_op How(spin_system& sys);
```

**Description:**

The function ***How*** returns a general operator containing the standard isotropic liquid NMR Hamiltonian in belonging to the spin system specified by the argument. The operator returned has frequency units and is computed by the formula

$$\boldsymbol{H}_o = \boldsymbol{H}_{cs} + \boldsymbol{H}_{Jw}, \tag{9-6}$$

where $\boldsymbol{H}_{cs}$ is the isotropic chemical shift Hamiltonian in the (multiply) rotating frame and $\boldsymbol{H}_{Jwh}$ isotropic scalar coupling Hamiltonian which implicitly sets heteronuclear couplings as being weak.

**Return Value:**

Isotropic liquid NMR Hamiltonian as a general operator in frequency units.

**Example(s):**

```
#include <gamma.h>
spin_system(ab);                    //
gen_op Ham;                         //
Ham = Ho(ab);                       //
```

**See Also:**

## 8.6.3 Ho_lab

**Usage:**

```
#include <nmr_ham.h>
gen_op Ho_lab(spin_system& sys);
```

**Description:**

The function ***Ho_lab*** returns a general operator containing the standard isotropic liquid NMR Hamiltonian in the laboratory frame belonging to the spin system specified by the argument. The operator returned has frequency units and is computed by the formula

$$\boldsymbol{H}_o = \boldsymbol{H}_{cslab} + \boldsymbol{H}_{Jwh},$$

where $\boldsymbol{H}_{cslab}$ is the isotropic chemical shift Hamiltonian in the laboratory frame and $\boldsymbol{H}_{Jwh}$ isotropic scalar coupling Hamiltonian which implicitly sets heteronuclear couplings as being weak.

**Return Value:**

Isotropic liquid NMR Hamiltonian as a general operator in frequency units.

**Example(s):**

```
#include <gamma.h>
spin_system(ab);                    //
gen_op Ham;                         //
Ham = Ho(ab);                       //
```

**See Also:**

## 8.6.4 Hcs

**Usage:**

```
#include <gamma.h>
gen_op Hcs(spin_system& sys);
```

**Description:**

The function ***Hcs*** returns a general operator containing the isotropic chemical shift Hamiltonian belonging to the spin system ***sys*** specified by the argument. The operator returned has frequency units and is computed by the formula

$$\boldsymbol{H}_{cs} = \sum_{i=1}^{spins} -\omega_i \boldsymbol{I}_{iz},$$

where for spin i, $\nu_i$ is the isotropic chemical shift (in Hertz) and $I_{iz}$ the z-component of spin angular momentum (taken as unitless).

## Return Value:

Isotropic chemical shift Hamiltonian as a general operator in frequency units.

## Example:

```
#include <gamma.h>
spin_system sys(2);                    // A 2 spin system (default 2 protons)
sys.shift(0, 100);                     // Set 1st spin isotropic shift to 100 Hz
sys.shift(1, 223.5);                   // Set 2nd spin shift to 100 Hz
gen_op H = Hcs(sys);                   // Get the isotropic shift Hamiltonian
```

## Mathematical Basis:

This chemical shift Hamiltonian blends the Zeeman Hamiltonian with the isotropic (rotationally invariant) component of chemical shielding Hamiltonian and general offset terms to account for any reference frequency or frequencies. In frequency units, this is

$$\boldsymbol{H}_{cs} = \sum_{i=1}^{spins} H_i^Z + H_i^{CSI} + \Omega_{ref} \boldsymbol{I}_{zi} = \sum_{i}^{spins} -\omega_i \boldsymbol{I}_{zi}$$

where the static magnetic field is assumed to be along the positive z-axis. Note that the frequencies used in construction of the chemical shift hamiltonian are relative values. The larger the chemical shift (for nuclei with positive gyromagnetic ratios) the smaller the chemical shielding. The behavior of magnetization under the influence of the shift Hamiltonian will be **analogous to what is observed under the effect of the Zeeman Hamiltonian: magnetization associated with a positive shift value under the influence of the shift Hamiltonian will rotate clockwise**[1] when viewed from the +z axis down to the xy-plane. This is equivalent to working in a rotating frame at the Larmor frequency of the spins isotope type, as shown in the following figure[2].

[1] Again, this assumes a positive gyromagnetic ratio.

[2] The plot of magnetization versus time was generated by the GAMMA program HShift.cc found at the end of this Chapter.

***Magnetization Response to the Shift Hamiltonian***

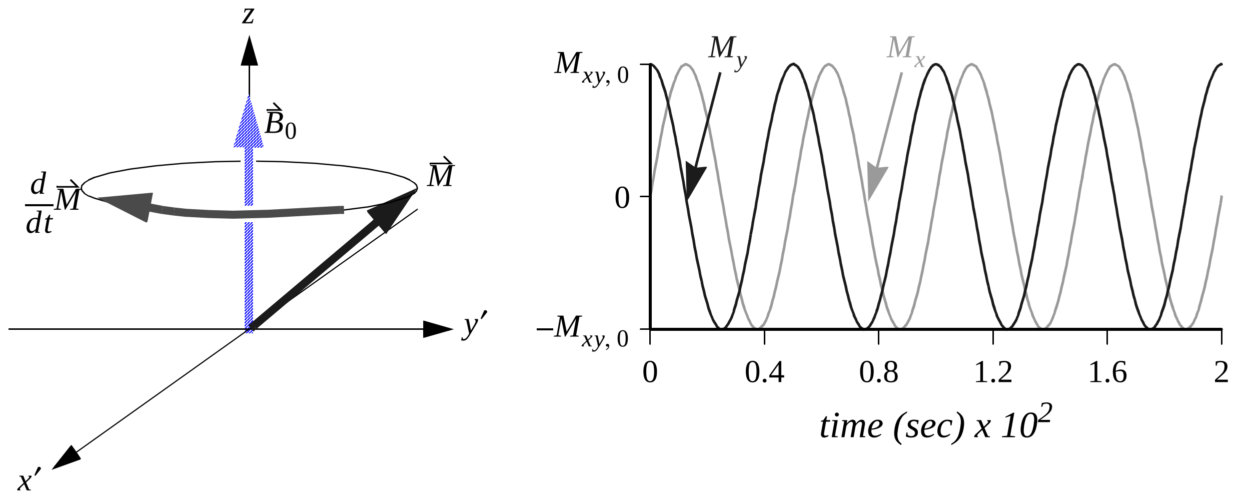


*Figure 4-1* - Evolution of magnetization under the effects of the shift Hamiltonian. The figures are shown are for a single spin having a chemical shift of 200 Hz relative to an unspecified rotating frame following a pulse applied along the x’ axis of 45 degrees. The figure on the left is constructed from classical arguments based on the cross product between $\vec{M}$ and $\vec{B}_0$. The figure on the right shows magnetization components of a single (unshifted) proton evolving in time in the rotating frame.

**See Also: Hcs_lab, Hz**

## 8.6.5 Hcs_lab

**Usage:**

```
#include <gamma.h>
gen_op Hcs_lab(spin_system& sys);
```

**Description:**

The function ***Hcs_lab*** returns a general operator containing both the isotropic chemical shift Hamiltonian and the Zeeman Hamiltonian associated with the spin system ***sys*** specified by the argument. The operator returned has frequency units and is computed by the formula

$$\boldsymbol{H}_{cs\ lab} = -\sum_{i}^{spins} (\Omega_{i,\,ref} + \omega_i)\boldsymbol{I}_{zi}$$

where the static magnetic field is assumed to be along the positive z-axis. Note that the frequencies used in construction of the chemical shift part of the Hamiltonian are relative values. The reference frequency, for isotopes with a positive gyromagnetic ratio, will be a large positive value. The behavior of magnetization under the influence of the shift Hamiltonian in the laboratory frame will be analogous to what is observed under the effect of the Zeeman Hamiltonian: **magnetization associated with a positive gyromagnetic ratio will**

**rotate clockwise** when viewed from the +z axis down to the xy-plane.

**Return Value:**

Isotropic chemical shift Hamiltonian as a general operator in frequency units.

**Example(s):**

```
#include <gamma.h>
spin_system(ab);
gen_op Ham;
Ham = Hcs_lab(ab);
```

**Mathematical Basis:**

This Hamiltonian is the isotropic (rotationally invariant) component of the total chemical shift Hamiltonian added to the Zeeman Hamiltonian. In energy units, these are

$$\boldsymbol{H}_{cs\ lab} = \sum_{i=1}^{spins} H_i^Z + H_i^{CSI} = \sum_{i}^{spins} -\omega_i\boldsymbol{I}_{zi} - \Omega_{ref}\boldsymbol{I}_{zi} = -\sum_{i}^{spins} (\Omega_{i,\,ref} + \omega_i)\boldsymbol{I}_{zi}$$

where for spin $i$, $\nu_i$ is its chemical shift (in Hertz), $\Omega_i$ the Larmor frequency of the spin isotope in Hertz, and $I_{iz}$ the z-component of spin angular momentum (taken as unitless).

**See Also: Hcs, Hz**

## 8.6.6 HJ

**Usage:**

```
#include <gamma.h>
gen_op HJ (spin_system& sys);
```

**Description:**

The function ***HJ*** returns the isotropic scalar coupling Hamiltonian as given by.

$$\boldsymbol{H}_J = \sum_{i}^{spins}\sum_{j>i}^{spins} J_{ij}\boldsymbol{I}_i \bullet \boldsymbol{I}_j = \begin{array}{c} \displaystyle\sum_{i}^{spins}\sum_{j>i}^{spins} J_{ij}(\boldsymbol{I}_{iz}\boldsymbol{I}_{jz} + \boldsymbol{I}_{ix}\boldsymbol{I}_{jx} + \boldsymbol{I}_{iy}\boldsymbol{I}_{jy}) \\ \displaystyle\sum_{i}^{spins}\sum_{j>i}^{spins} J_{ij}(\boldsymbol{I}_{iz}\boldsymbol{I}_{jz} + \boldsymbol{I}_{i+}\boldsymbol{I}_{j+} + \boldsymbol{I}_{i-}\boldsymbol{I}_{j-}) \end{array} . \qquad (9\text{-}7)$$

A general operator is returned with frequency (Hertz) units. All scalar coupling constants are obtained from the current values in the spin system given as an argument.

**Return Value:**

Isotropic strong scalar coupling Hamiltonian is returned as a general operator.

**Example(s):**

```
#include <gamma.h>
spin_system(ab);                        //
gen_op Ham;                             //
Ham = HJ(ab);                           //
```

**Mathematical Basis:**

The scalar coupling Hamiltonian can be defined as the product of ******.

**See Also: HJw, HJwh, Hcs, Hiso**

## 8.6.7 HJw

**Usage:**

```
#include <gamma.h>
gen_op HJw(spin_system&);
```

**Description:**

The function ***HJw*** returns the “weak” isotropic scalar coupling Hamiltonian as given by the formula

$$\boldsymbol{H}_{Jw} = \sum_{i}^{spins} \sum_{j>i}^{spins} J_{ij}\boldsymbol{I}_{iz}\boldsymbol{I}_{jz} . \tag{9-8}$$

This is sometimes referred to as neglecting the non-secular terms of the scalar coupling Hamiltonian. Note that there is no distinction made between homonuclear and heteronuclear coupling term in this treatment.All terms contain only the diagonal elements.

**Return Value:**

Weak coupling term as a general operator.

**Example(s):**

```
#include <gamma.h>
spin_system(ab);                        //
gen_op Ham;                             //
Ham = HJw(ab);                          //
```

**See Also:**

## 8.6.8 HJwh

**Usage:**

```
#include <gamma.h>
gen_op HJwh(spin_system&);
```

**Description:**

The function HJwh returns the scalar coupling part of the Hamiltonian in which all homonuclear couplings are strong and all heteronuclear ones are weak.

$$\boldsymbol{H}_{Jwh} = \sum_{i}^{spins}\sum_{j>i}^{spins} J_{ij}\{\boldsymbol{I}_{iz}\boldsymbol{I}_{jz} + \delta_{\gamma_i\gamma_j}(\boldsymbol{I}_{ix}\boldsymbol{I}_{jx} + \boldsymbol{I}_{iy}\boldsymbol{I}_{jy})\} \tag{9-9}$$

This function should be used for simulations done in multiply rotating frames. Note that the isotopes of the spin system should be specified prior to this function call.A general operator is returned with frequency (Hertz) units.

## Return Value:

Coupling term as a general operator.

## Example(s):

```
#include <gamma.h>
spin_system(ab);                    //
gen_op Ham;                         //
Ham = HJwh(ab);                     //
```

## See Also:

### 8.6.9 Hz

**Usage:**

```
#include <gamma.h>
gen_op Hz(spin_system& sys);
```

**Description:**

The function ***Hz*** returns a general operator containing the Zeeman Hamiltonian associated with the spin system ***sys*** specified by the argument. The returned operator has units of Hertz and is computed by the formula

$$\boldsymbol{H}_Z = \sum_i^{spins} -\Omega_{i,o}\boldsymbol{I}_{iz}$$

with $\Omega_{i,o}$ is a spectrometer frequency of spin $i$ and $I_{iz}$ its z-component of spin angular momentum (taken as unitless).

**Return Value:**

A general operator in units of Hz.

**Example:**

```
#include <gamma.h>
spin_system sys(2);                    // A 2 spin system (default 2 protons)
sys.Omega(500.0);                      // Set static field to 500 MHz
gen_op H = Hz(sys);                    // Get the Zeeman Hamiltonian
```

**Mathematical Basis:**

The Zeeman Hamiltonian is isotropic (rotationally invariant), defined in energy units as

$$\boldsymbol{H}_Z = \sum_i^{spins} -h\gamma_i B_o \vec{\boldsymbol{I}}_i \bullet \vec{\boldsymbol{B}}_n = \sum_i^{spins} -h\gamma_i B_o \boldsymbol{I}_{zi} = \sum_i^{spins} -h\Omega_{i,o}\boldsymbol{I}_{iz}$$

where the static magnetic field in GAMMA is taken to be along the positive z-axis. The frequency $\Omega_{i,o}$ is roughly the Larmor frequency of the spin and taken to be the same for all spins of the same isotope type as defined by $\gamma_i B_o = \Omega_{i,o}$ where the gyromagnetic ratio $\gamma_i$ is appropriate for the spin.

$$\boldsymbol{H}_Z^{\text{homonuclear}} = -h\Omega_o \sum_i^{spins} \boldsymbol{I}_{zi} = -h\Omega_o F_z$$

Variations of this formula depend on whether the Hamiltonian is written in energy, angular frequency, or linear frequency units[1]. The function ***Hz*** returns the Hamiltonian in units of Hertz, thus a rigorous formula is

[1] The sign on the Zeeman Hamiltonian may vary in the literature if the static magnetic field is defined to be along the -z axis rather than +z axis as it is in GAMMA.

$$\boldsymbol{H}_Z = \frac{1}{2\pi}\sum_{i=1}^{spins} -\gamma_i B_o \boldsymbol{I}_{zi}$$

Note that general effect of the Zeeman Hamiltonian in GAMMA is to cause magnetization to **precess in a clockwise direction**[1] when viewed from the +z axis down to the xy-plane. From a classical standpoint we know that any magnetization $\vec{M}$ interacting with a magnetic field $\vec{B}$ will experience a torque proportional to

$$\frac{d}{dt}\vec{M} \propto \vec{M} \times \vec{B}$$

Thus, for magnetization under the influence of the static field $\vec{B}_0$ along the +z axis we conclude using the right hand rule that any magnetization should precess in a clockwise fashion about the z-axis as shown in the following figure[2].

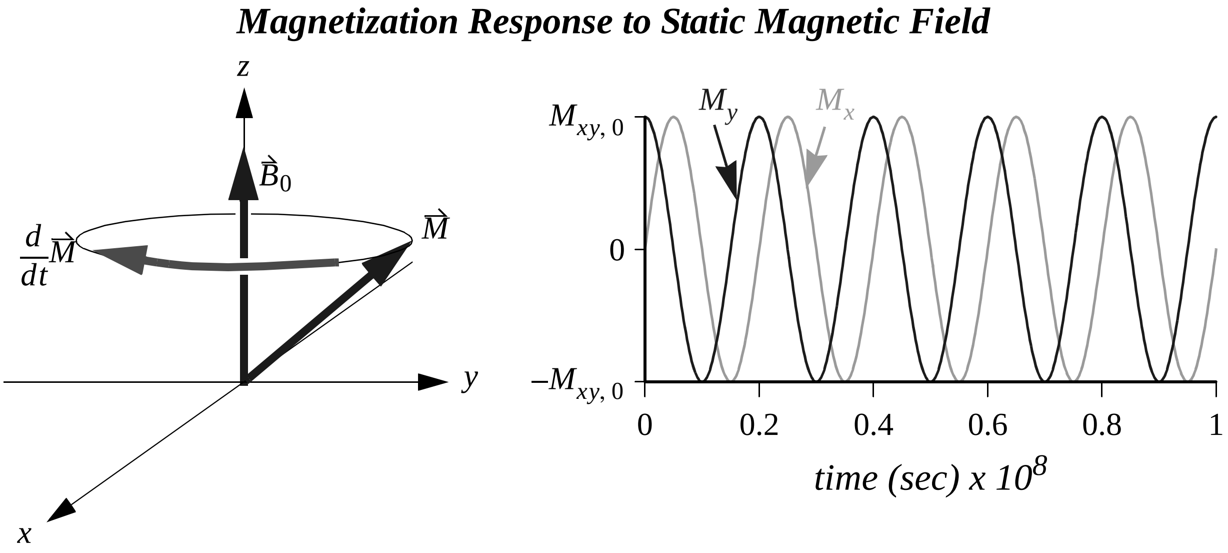


*Figure 4-2* - Evolution of magnetization under the effects of the Zeeman interaction. The figure on the left is construed from classical arguments based on the cross product between $\vec{M}$ and $\vec{B}_0$. The figure on the right shows magnetization components of a single (unshifted) proton evolving in time under a field strength of 500 MHz. The spin was initial pulsed so that its initial magnetization was 45 degrees down from the +z axis.

**See Also: Hcs, Hcs_lab**

[1] This assumes a positive gyromagnetic ratio. For isotopes with a negative $\gamma$ the Zeeman precession will be counter-clockwise when looking down the +z axis to the xy-plane.

[2] The plot of magnetization versus time was generated by the GAMMA program HZeeman.cc found at the end of this Chapter.

## 8.6.10 H1

### Usage:

```
#include <gamma.h>
gen_op H1(spin_system&, char* iso, double gamB1=2.5e4, double phi=0.0);
```

### Description:

The function ***H1*** returns the scalar coupling part of the Hamiltonian in which all homonuclear couplings are strong and all heteronuclear ones are weak.

$$\boldsymbol{H}_1 = \sum_{i}^{spins} \sum_{j>i}^{spins} J_{ij}\{\boldsymbol{I}_{iz}\boldsymbol{I}_{jz} + \delta_{\gamma_i\gamma_j}(\boldsymbol{I}_{ix}\boldsymbol{I}_{jx} + \boldsymbol{I}_{iy}\boldsymbol{I}_{jy})\} \tag{9-10}$$

This function should be used for simulations done in multiply rotating frames. Note that the isotopes of the spin system should be specified prior to this function call.A general operator is returned with frequency (Hertz) units.

### Return Value:

Coupling term as a general operator.

### Example(s):

```
#include <gamma.h>
spin_system(ab);                    //
gen_op Ham;                         //
Ham = HJwh(ab);                     //
```

### See Also:

## 8.6.11 Heff

### Usage:

```
#include <gamma.h>
gen_op Heff(spin_system& sys, gen_op& Ho, String& iso, double Wrf=0,
                                                     double gamB1=2.5e4, double phi=0.0);
```

### Description:

The function ***Heff*** returns the effective Hamiltonian for the spin system ***sys*** when there exists the combined effects of a static Hamiltonian ***Ho*** and an applied rf-field. The field is of frequency ***Wrf***, strength ***gamB1***, phase ***phi***, affecting isotopes of type ***iso***. The effective Hamiltonian formally defined in the rotating frame of the rf-field by

$$\boldsymbol{H}_{eff} = \boldsymbol{H}_o + \Omega_{rf}\boldsymbol{F}_{z,\{i\}} - \gamma B_1 \boldsymbol{F}_{x,\varphi,\{i\}} \tag{9-11}$$

The arguments ***Ho***, ***Wrf***, and ***gamB1*** should be in the same units, normally Hertz, and set the units of the returned operator.where $\boldsymbol{H}_o$ is a general operator, usually the static isotropic Hamiltonian. The output Hamiltonian, as well as $\boldsymbol{H}_o$, The term $\Omega_{rf}\boldsymbol{F}_{z,\{i\}}$ shifts the static Hamiltonian in a rotating frame referenced to the same zero as the rf-field frequency, $\Omega_{rf}$. Both are input in Hertz, by the arguments ***Ho*** and ***Wrf*** respectively. The spin system ***sys*** should contain at least one spin of the isotope type specified by ***iso***, the applied field is assumed to be close to the Larmor frequency of this spin type. The field strength ***gamB1*** has a default value which would set up a 90 pulse on proton if the field is on for 10 μsec. The phase ***phi*** is input in degrees. *Note: The field frequency required is usually the opposite sign as one might suppose initially. Consider a single spin system with a shift of 100 Hz. If* ***Ho*** *is in the rotating frame the spin (shielded) will oscillate at a frequency which is -100 Hz, i.e. the negative of the shift value! Then the required* ***Wrf*** *is -100 Hz as well.*

## Return Value:

A general operator.

## Example:

```
#include <nmr_ham.h>
spin_system sys;                          // Declare a spin system
sys.read(“filename”);                     // Read the system from file filename
gen_op H0 = Ho(sys);                      // Generate the static isotropic Hamiltonian
double Wrf = -100.0;                      // Set the field frequency
double gamB1 = 1000.0;                    // Set the field strength
String iso = sys.symbol(0);               // Get first spin isotope type
gen_op He = Heff(sys,H0,iso,Wrf, gamB1);  // Get the effective Hamiltonian
```

## Mathematical Basis:

The effective Hamiltonian is defined in the rotating frame of the rf-field by

$$\boldsymbol{H}_{eff} = \boldsymbol{H}_o + \Omega_{rf}\boldsymbol{F}_{z,\{i\}} - \gamma_i B_1 \boldsymbol{F}_{x,\varphi,\{i\}} \tag{9-12}$$

where $\boldsymbol{H}_o$ is a general operator, usually the static isotropic Hamiltonian. The term $\Omega_{rf}\boldsymbol{F}_{z,\{i\}}$ shifts the static Hamiltonian into a rotating frame about the z-axis at rf-field frequency, $\Omega_{rf}$. This shift affects only spins of a specific isotope type $\{i\}$, leaving the any others in their original frame. The frequency $\Omega_{rf}$ should be specified relative to any current rotating frame of spins $\{i\}$ in $\boldsymbol{H}_o$. The term $-\gamma B_1\boldsymbol{F}_{x,\varphi,\{i\}}$ accounts for the applied field Hamiltonian, time-independent in the rotating frame at $\Omega_{rf}$. The field is assumed to affect only spins $\{i\}$ of the same isotope type, and is applied with phase angle $\varphi$ relative to the x-axis.

For small applied field strengths (usually much smaller that the static field associated with $\boldsymbol{H}_o$), the effective Hamiltonian is dominated by the static isotropic Hamiltonian in the rotating frame of the rf-field, $\Omega_{rf}$. As the field strength, $B_1$, is increased, the $\boldsymbol{F}_x$ term will dominate this Hamiltonian. The following plot depicts how the four eigenvalues of $\boldsymbol{H}_{eff}$ in a two spin system change as the field strength is turned on[1].

[1] The source code which produced the plot is given at the end of this chapter named Heff.cc.

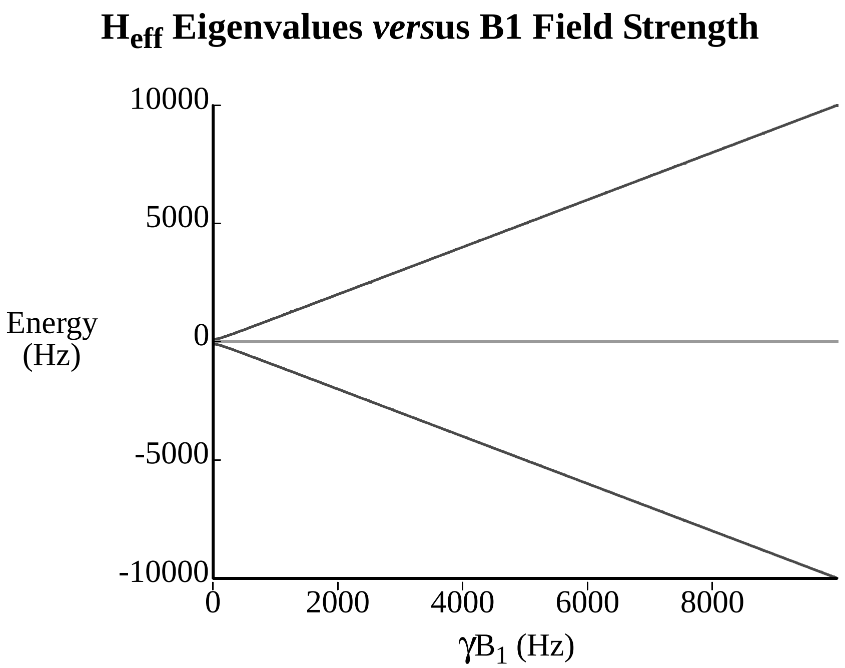


*Figure 4-3* The four eigenvalues of $\boldsymbol{H}_{eff}$ relative to the applied field strength.

**See Also:**

## 8.7 Description

The density matrix equation of motion (von Neumann, Liouville) is

$$ih\frac{d\sigma}{dt} = [\boldsymbol{H}, \sigma] \tag{4-1}$$

where $\boldsymbol{H}$ is the acting Hamiltonian. It is thus the active Hamiltonian which dictates the evolution of the density matrix in time and motivates this discussion.

### 8.7.1 Time Independent Hamiltonians

It is easy to verify that the solution to the Liouville equation under a time independent Hamiltonian is

$$\sigma(t+t_0) = e^{(-i\boldsymbol{H}t)/h}\sigma(t_0)e^{(i\boldsymbol{H}t)/h} \qquad . \tag{4-2}$$

Here t is the time during which the Hamiltonian has acted on the spin system, $\sigma(t_0)$ represents the initial state of the system and $\sigma(t+t_0)$ the final state. It is common to write this in terms of the propagator $\boldsymbol{U}$,

$$\sigma(t+t_0) = \boldsymbol{U}\sigma(t_0)\boldsymbol{U}^{-1} \qquad \text{where} \qquad \boldsymbol{U} = e^{-i\boldsymbol{H}t/h} \tag{4-3}$$

The equivalent solution to equation (4-1), *i.e.* the solution to the von Neumann equation under a time independent Hamiltonian, can be written with superoperator formalism as[1]

$$\sigma(t+t_0) = e^{-i\boldsymbol{H}t}\sigma(t_0)e^{i\boldsymbol{H}t} = \boldsymbol{U}\sigma(t_0)\boldsymbol{U}^{-1} = \hat{\boldsymbol{U}}\sigma(t_0) \qquad . \tag{4-4}$$

In this context, the superoperator equivalent of the propagator $\boldsymbol{U}$, namely $\hat{\boldsymbol{U}}$, is determined from[2]

$$\hat{\boldsymbol{U}} = \boldsymbol{U} \otimes \boldsymbol{U}^* \tag{4-5}$$

Here $\boldsymbol{U}^*$ is the complex conjugate of the propagator $\boldsymbol{U}$ and $\otimes$ denotes a tensor product.

We now consider the application of these formula to the treatment of NMR. In NMR the dominating contribution to the Hamiltonian is the Zeeman term, $\boldsymbol{H}_Z$, which is time independent. In a homonuclear spin system this is (as will be shown later in this Chapter)

$$\boldsymbol{H}^Z = -\Omega_o\boldsymbol{F}_z$$

Under the Zeeman Hamiltonian, equation (4-2) then becomes

$$\sigma(t+t_0) \approx e^{i\boldsymbol{F}_z\Omega_o t}\sigma(t_0)e^{-i\boldsymbol{F}_z\Omega_o t} = \boldsymbol{R}_z^{-1}(\Omega t)\sigma(t_0)\boldsymbol{R}_z(\Omega t) \tag{4-6}$$

[1] See EBW, page 16, equation (2.1.41)
[2] See EBW, page 24, equation (2.1.83).

Evidently, the general effect of the applied static field (Zeeman) is to cause some rotation about the z-axis (the axis of the static field), and at common field strengths this rotation will be in hundreds of Megahertz.

## 8.7.2 The Rotating Frame

Although the Zeeman interaction is the largest in NMR it is not of direct importance. Effects due to smaller contributions to the NMR Hamiltonian are more interesting than the Megahertz oscillations due to the Zeeman term. It is for this reason that, both theoretically and experimentally, one works in a rotating frame (or multiple rotating frames). Viewed from a coordinate system rotating about the z-axis at a frequency near the Zeeman induced precession, such Zeeman oscillations are either absent or slow.

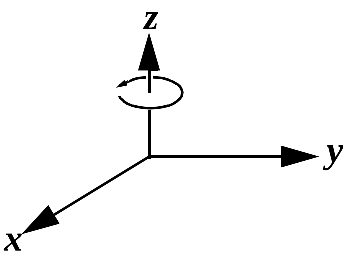
The shift into the rotating frame mathematically involves rotations of all operators. For a rotating frame about the axis $u$ this is

$$\underset{\sim}{Op} = \boldsymbol{R}_u(\Omega t) Op \boldsymbol{R}_u^{-1}(\Omega t) \tag{4-7}$$

and the Liouville equation in the rotating frame takes the form

$$ih\frac{d}{dt}\underset{\sim}{\sigma} = [\underset{\sim}{\boldsymbol{H}} + \Omega F_u, \underset{\sim}{\sigma}] \tag{4-1}$$

where $F_u$ is the component of angular momentum along the axis of rotation.



**Mathematical Basis:**

## 8.7.3 The Zeeman Hamiltonian

The Zeeman Hamiltonian accounts for the interaction between magnetic moment (a nuclear spin) and an applied static magnetic field $\vec{\boldsymbol{B}}_o$. The classical interaction energy between an applied field and a nuclear spin is

$$E_i^Z = -\vec{\mu}_i \bullet \vec{\boldsymbol{B}} \tag{4-2}$$

where $\vec{\mu}$ is the magnetic moment, $i$ the spin index, $E$ the energy, and superscript $Z$ used to denote Zeeman. The associated Hamiltonian is obtained from substitution of $h\gamma\vec{\boldsymbol{I}}_i$ for $\vec{\mu}_i$ and we also use $\vec{\boldsymbol{B}}_o$ instead of $\vec{\boldsymbol{B}}$ so that the equation is in standard NMR nomenclature.

$$\boldsymbol{H}_i^Z = -h\gamma_i\vec{\boldsymbol{I}}_i \bullet \vec{\boldsymbol{B}}_o \tag{4-3}$$

In matrix form this equation looks like

$$\boldsymbol{H}_i^Z = -h\gamma_i\begin{bmatrix}\boldsymbol{I}_{ix} & \boldsymbol{I}_{iy} & \boldsymbol{I}_{iz}\end{bmatrix} \bullet \begin{bmatrix}B_{0x}\\ B_{0y}\\ B_{0z}\end{bmatrix}. \tag{4-4}$$

Taking the magnitude of the applied field out, equation (4-22) is simply

$$\boldsymbol{H}_i^Z = -h\gamma_i B_o \sum_{v}^{axes} \langle 1|\vec{\boldsymbol{I}}_i|v\rangle\langle v|\vec{\boldsymbol{B}}_n|1\rangle \tag{4-5}$$

with $v \in \{x, y, z\}$ and $\vec{\boldsymbol{B}}_n$ a normalized magnetic field vector in the direction of the applied field.

### Rank 1 Zeeman Hamiltonian Treatment

Letting

$$\boldsymbol{T}_i^Z = \begin{bmatrix}\boldsymbol{I}_{ix} & \boldsymbol{I}_{iy} & \boldsymbol{I}_{iz}\end{bmatrix} \qquad \text{and} \qquad \boldsymbol{A}_i^Z = \begin{bmatrix}B_{0x}\\ B_{0y}\\ B_{0z}\end{bmatrix}_i \tag{4-6}$$

the Zeeman Hamiltonian be placed into the context of a scalar product of two rank 1 Cartesian tensors (vectors).

$$\boldsymbol{H}_i^Z = -h\gamma_i B_o \boldsymbol{T}_i^Z \bullet \boldsymbol{A}_i^Z$$

or (4-7)

$$\boldsymbol{H}_i^Z = -h\gamma_i B_o \sum_{u}^{axes} \langle 1|\boldsymbol{A}_i^Z|u\rangle\langle u|\boldsymbol{T}_i^Z|1\rangle$$

Rewriting equation (4-7) in terms of 4 irreducible spherical rank 1 tensor components produces

$$\boldsymbol{H}_i^Z = -h\gamma_i B_o \sum_{l=0}^{1} \sum_{m}^{\pm l} (-1)^m \boldsymbol{A}_{l,-m}^{Z1}(i) \bullet \boldsymbol{T}_{lm}^{Z1}(i) \qquad . \tag{4-8}$$

We obtain the 4 irreducible spherical components of the Zeeman rank 1 spin tensor directly from the Cartesian components, $\langle v|\hat{\boldsymbol{T}}_i|u\rangle$ , as indicated in GAMMA Class Documentation on Spin Tensor. These are

$$\boldsymbol{T}_{l,m}^{Z1}(i) \quad ,$$

where $Z$ signifies the Zeeman interaction, and $i$ is the spin index. The tensor index $l$ spans the rank: $l \in [0, 1]$ while the tensor index $m$ spans $l$: $m \in [-l, l]$ The four formulas for these quantities a listed in the following figure, they are valid for any chosen spatial axes.

### Rank 1 Irreducible Spherical Spin Tensor Components

$$T_{0,0}(i) = 0 \qquad T_{1,0}(i) = I_{iz} \qquad T_{1,\pm 1}(i) = \frac{\mp 1}{\sqrt{2}} I_{i\pm}$$

*Figure 4-4* : Components of the complete spin tensor in the rank 1 Zeeman Hamiltonian treatment.

For a single spin 1/2 particle the Zeeman spin tensor components are, in the Hilbert space of the spin itself, given in the following figure (the spin index is implicit here, and these are again valid for any chosen spatial axes)

### Rank 1 Irreducible Spherical Spin Tensor Component Matrix Representations

$$T_{0,0}^{(1)} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \qquad T_{1,0}^{(1)} = \frac{1}{2}\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \qquad T_{1,-1}^{(1)} = \frac{1}{\sqrt{2}}\begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix} \qquad T_{1,1}^{(1)} = \frac{-1}{\sqrt{2}}\begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}$$

*Figure 4-5* : Matrix representations in Hilbert space of the rank 1 spin tensor components for a single spin I=1/2 particle.

The 4 irreducible spherical components of the Zeeman spatial tensor (rank 1) are formally specified with the nomenclature

$$A^{Z1}_{l,m}(i) \quad ,$$

where again the subscript *l* spans the rank as *l* = [0, 1], and the subscript *m* spans *+/- l, m* = [-*l*, *l*]. The four formulas which produce these quantities are

### *Rank 1 Zeeman Irreducible Spherical Spatial Tensor Components*

$$\boxed{A^{Z1}_{0,0}(i) = 0 \qquad A^{Z1}_{1,0}(i) = A^{Z1}_{zz}(i) = B_{nz} \qquad A^{Z1}_{1,\pm1}(i) = \mp A^{Z1}_{\pm} = B_{nx} \pm iB_{ny}}$$

*Figure 4-6* : Components of the spatial tensor in the rank 1 Zeeman Hamiltonian treatment.

If we express the spatial tensor (the field vector) in the laboratory axes, and the normalized vector is given by $\vec{\boldsymbol{B}}_n = \hat{k}$, we have

### *Field Aligned Rank 1 Zeeman Irreducible Spherical Spatial Tensor Components*

$$\boxed{A^{Z1}_{0,0}(i, LAB) = 0 \qquad A^{Z1}_{1,0}(i, LAB) = 1 \qquad A^{Z1}_{1,\pm1}(i, LAB) = 0}$$

*Figure 4-7* : Components of the spatial tensor in the rank 1 Zeeman Hamiltonian treatment when the static field is aligned along the +z axis.

If we define the Zeeman interaction constant to be

$$\xi^Z_i = -h\gamma_i B_o \tag{4-9}$$

the equation for the Zeeman Hamiltonian becomes (we can remove the $l = 0$ contribution)

$$\boldsymbol{H}^Z_i(AAS) = \xi^Z_i \sum_m^{\pm1} (-1)^m \boldsymbol{A}^{Z1}_{1,-m}(i, AAS) \bullet \boldsymbol{T}^{Z1}_{1,m}(i) \qquad . \tag{4-10}$$

where the spatial tensor components in an arbitrary axis system are related to those in the laboratory frame by the rotation formula

$$A^{Z1}_{1,m}(i, AAS) = \sum_{m'}^{\pm1} D^1_{mm'}(\Omega) A^{Z1}_{1,m'}(i, LAB)$$

Normally it is the laboratory frame in which we wish our Hamiltonian expressed so then we have

$$\boldsymbol{H}^Z_i(LAB) = \xi^Z_i \boldsymbol{A}^{Z1}_{1,0}(i, LAB) \bullet \boldsymbol{T}^{Z1}_{1,0}(i) = -h\gamma_i B_o I_{iz} \qquad . \tag{4-11}$$

Summing over all spins in a spin system, the total Zeeman Hamiltonian is then

$$\boldsymbol{H}_i^Z(LAB) = \sum_i^{spins} -h\gamma_i B_o I_{iz} = \sum_i^{spins} -h\Omega_{i,o} I_{iz} \quad . \tag{4-12}$$

## Rank 2 Zeeman Hamiltonian Treatment

We could choose to treat the Zeeman Hamiltonian in terms of rank 2 tensors, although there is no more insight to be gained over the rank 1 treatment. By insertion of the identity matrix we have

$$\boldsymbol{H}_i^Z = -h\gamma_i B_o \sum_v^{axes} \langle 1|\vec{\boldsymbol{I}}_i|v\rangle\langle v|\vec{\boldsymbol{B}}_n|1\rangle = -h\gamma_i B_o \sum_u^{axes} \sum_v^{axes} \langle 1|\vec{\boldsymbol{I}}_i|u\rangle\langle u|1|v\rangle\langle v|\vec{\boldsymbol{B}}_n|1\rangle$$

$$\boldsymbol{H}_i^Z = -h\gamma_i B_o \begin{bmatrix} \boldsymbol{I}_{ix} & \boldsymbol{I}_{iy} & \boldsymbol{I}_{iz} \end{bmatrix} \bullet \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \bullet \begin{bmatrix} B_{nx} \\ B_{ny} \\ B_{nz} \end{bmatrix}$$

The identity matrix being the first rank 2 tensor, a second tensor is formed from the dyadic product of the two vectors, explicitly done *via*

$$\begin{bmatrix} B_{nx} \\ B_{ny} \\ B_{nz} \end{bmatrix} \bullet \begin{bmatrix} \boldsymbol{I}_{ix} & \boldsymbol{I}_{iy} & \boldsymbol{I}_{iz} \end{bmatrix} = \begin{bmatrix} B_{nx}\boldsymbol{I}_{xi} & B_{nx}\boldsymbol{I}_{yi} & B_{nx}\boldsymbol{I}_{zi} \\ B_{ny}\boldsymbol{I}_{xi} & B_{ny}\boldsymbol{I}_{yi} & B_{ny}\boldsymbol{I}_{zi} \\ B_{nz}\boldsymbol{I}_{xi} & B_{nz}\boldsymbol{I}_{yi} & B_{nz}\boldsymbol{I}_{zi} \end{bmatrix} .$$

The Zeeman Hamiltonian can thus be formulated as a scalar product of two rank 2 tensors. Letting $\hat{\boldsymbol{T}}_i = \vec{\boldsymbol{B}}_n \vec{\boldsymbol{I}}_i$, we have

$$\boldsymbol{H}_i^Z = -h\gamma_i B_o \boldsymbol{1} \bullet \hat{\boldsymbol{T}}_i^{Z2}$$

or equivalently (4-13)

$$\boldsymbol{H}_i^Z = -h\gamma_i B_o \sum_u^{axes} \sum_v^{axes} \langle u|\boldsymbol{1}|v\rangle\langle v|\boldsymbol{T}_i^{Z2}|u\rangle$$

Rewriting equation (4-13) in terms of irreducible spherical components rather than the current Cartesian components

$$\boldsymbol{H}_i^Z = -h\gamma_i B_o \sum_{l=0}^{2} \sum_m^{\pm l} (-1)^m A_{l,m}^{Z2}(i) \bullet \boldsymbol{T}_{l,-m}^{Z2}(i) \tag{4-14}$$

We can obtain the 9 irreducible spherical components of the Zeeman rank 2 “spin” tensor[1] directly from the Cartesian components, $\langle v|\boldsymbol{T}_i^{Z2}|u\rangle$ , as indicated in GAMMA Class Documentation on Spin Tensor. These are

$$\boldsymbol{T}_{l,m}^{Z2}(i) \quad ,$$

where $Z2$ signifies the rank 2 treatment of the Zeeman interaction, and $i$ is the spin index. The tensor index $l$ spans the rank: $l \in [0, 2]$ while the tensor index $m$ spans $l$: $m \in [-l, l]$ The nine formulas for these quantities a listed in the following figure where the field components are those of the normalized field vector $\vec{\boldsymbol{B}}_n$.[2]

### Zeeman Rank 2 Irreducible Spherical Spin-Space Tensor Components

$$T_{0,0}(i) = \frac{-1}{\sqrt{3}}\left[\boldsymbol{I}_{iz}B_z + \frac{1}{2}(\boldsymbol{I}_{i+}B_- + \boldsymbol{I}_{i-}B_+)\right] = \frac{-1}{\sqrt{3}}\vec{\boldsymbol{I}}_i \bullet \vec{\boldsymbol{B}}_n$$

$$T_{1,0}(i) = \frac{-1}{2\sqrt{2}}[\boldsymbol{I}_{i+}B_- - \boldsymbol{I}_{i-}B_+] \qquad T_{1,\pm1}(i) = \frac{-1}{2}[\boldsymbol{I}_{i\pm}\ B_z - \boldsymbol{I}_{iz}B_\pm\ ]$$

$$T_{2,0}(i) = \frac{1}{\sqrt{6}}[3\boldsymbol{I}_{iz}B_z - (\vec{\boldsymbol{I}}_i \bullet \vec{\boldsymbol{B}}_n)]$$

$$T_{2,\pm1}(i) = \mp\frac{1}{2}[\boldsymbol{I}_{i\pm}\ B_z + \boldsymbol{I}_{iz}B_\pm\ ] \qquad T_{2,\pm2}(i) = \frac{1}{2}[\boldsymbol{I}_{i\pm}\ B_\pm\ ]$$

*Figure 4-8* : Components of the “spin-space” tensor in the rank 2 Zeeman Hamiltonian treatment.

For a spin 1/2 particle and $\vec{B}_0 = B_0\vec{B}_n$ s, the matrix form of these tensor components are shown in the following figure in the single spin Hilbert space. The spin index has been omitted, the field components are those of the normalized vector $\vec{B}_n$.

[1] Due to the nature of the Zeeman interaction, the rank 2 tensor treatment produces a “spin” tensor $T^{Z2}$ which contains spatial components, namely the magnetic field vector. As a result, care must be used when performing spatial rotations on the rank 2 Zeeman tensors. Any spatial rotations must involve rotations of both $A^{Z2}$ and $T^{Z2}$

[2] For these formulae, it is important to note that it is the second component in the composite spin/space tensor which is set to the normalized magnetic field vector $\vec{B}_n$, 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. Our field vector has be set to point along the positive z-axis in the laboratory frame.

## Rank 2 Zeeman Spherical Spin-Space Tensor Component Matrix Representations

$$T^{(2)}_{0,0} = \frac{-1}{2\sqrt{3}}\begin{bmatrix} B_z & B_- \\ B_+ & -B_z \end{bmatrix} \qquad T^{(2)}_{1,0} = \frac{-1}{2\sqrt{2}}\begin{bmatrix} 0 & B_- \\ -B_+ & 0 \end{bmatrix} \qquad T^{(2)}_{1,-1} = \frac{-1}{2}\begin{bmatrix} -B_-/2 & 0 \\ B_z & B_-/2 \end{bmatrix} \qquad T^{(2)}_{1,1} = \frac{-1}{2}\begin{bmatrix} -B_+/2 & B_z \\ 0 & B_+/2 \end{bmatrix}$$

$$T^{(2)}_{2,0} = \frac{1}{2\sqrt{6}}\begin{bmatrix} 2B_z & -B_- \\ -B_+ & -2B_z \end{bmatrix} \quad T^{(2)}_{2,-1} = \frac{1}{2}\begin{bmatrix} B_-/2 & B_z \\ 0 & -B_-/2 \end{bmatrix} \quad T^{(2)}_{2,1} = \frac{-1}{2}\begin{bmatrix} B_+/2 & 0 \\ B_z & -B_+/2 \end{bmatrix} \quad T^{(2)}_{2,-2} = \frac{1}{2}\begin{bmatrix} 0 & 0 \\ B_- & 0 \end{bmatrix} \quad T^{(2)}_{2,2} = \frac{1}{2}\begin{bmatrix} 0 & B_+ \\ 0 & 0 \end{bmatrix}$$

*Figure 4-9* : Matrix representations in Hilbert space of the “spin-space” tensor components in the rank 2 Zeeman Hamiltonian treatment. Arrays shown are for a single spin I=1/2 particle.

The raising an lowering components of the field vector are defined in the standard fashion, namely $\vec{B}_{\pm} = B_x \pm iB_y$. The simplest situation occurs when magnetic field points along the positive z-axis, $\vec{\boldsymbol{B}}_n = \hat{\boldsymbol{k}}$, *i.e.* these spin-space tensors are written in the laboratory frame. Then, the (normalized) field vector simplifies, $\vec{B}_z = 1$ and $B_x = B_y = \vec{B}_{\pm} = 0$. The applicable equations for the shielding space-spin tensors are then as follows.

## Aligned Rank 2 Zeeman Irreducible Spherical Spin-Space Tensor Components

$$T_{0,0}(ij) = \frac{-1}{\sqrt{3}}\boldsymbol{I}_{iz}$$

$$T_{1,0}(ij) = 0 \qquad T_{1,\pm1}(ij) = \frac{-1}{2}\boldsymbol{I}_{i\pm}$$

$$T_{2,0}(ij) = \frac{2}{\sqrt{6}}\boldsymbol{I}_{iz}$$

$$T_{2,\pm1}(ij) = \mp\frac{1}{2}\boldsymbol{I}_{i\pm} \qquad T_{2,\pm2}(ij) = 0$$

*Figure 4-10* : Components of the “spin-space” tensor in the rank 2 Zeeman Hamiltonian treatment when the static field is aligned along the +z axis.

For a spin 1/2 particle and $\vec{B}_0 = B_0\vec{B}_n$ along the positive z-axis, the matrix form of these tensor components are shown in the following figure[1] (in the single spin Hilbert space).

[1] The GAMMA program which produced these matrix representations can be found at the end of this Chapter, Rank2SS_SpinT.cc.

### *Aligned Rank 2 Zeeman Spin-Space Tensor Component Matrix Representations*

$$T_{0,0}^{(2)} = \frac{-1}{2\sqrt{3}}\begin{bmatrix}1 & 0\\ 0 & -1\end{bmatrix} \qquad T_{1,0}^{(2)} = \begin{bmatrix}0 & 0\\ 0 & 0\end{bmatrix} \qquad T_{1,-1}^{(2)} = \frac{-1}{2}\begin{bmatrix}0 & 0\\ 1 & 0\end{bmatrix} \qquad T_{1,1}^{(2)} = \frac{-1}{2}\begin{bmatrix}0 & 1\\ 0 & 0\end{bmatrix}$$

$$T_{2,0}^{(2)} = \frac{1}{\sqrt{6}}\begin{bmatrix}1 & 0\\ 0 & -1\end{bmatrix} \qquad T_{2,1}^{(2)} = \frac{-1}{2}\begin{bmatrix}0 & 1\\ 0 & 0\end{bmatrix} \qquad T_{2,-1}^{(2)} = \frac{1}{2}\begin{bmatrix}0 & 0\\ 1 & 0\end{bmatrix} \qquad T_{2,-2}^{(2)} = \begin{bmatrix}0 & 0\\ 0 & 0\end{bmatrix} \qquad T_{2,2}^{(2)} = \begin{bmatrix}0 & 0\\ 0 & 0\end{bmatrix}$$

*Figure 4-11* : Matrix representations in Hilbert space of the “spin-space” tensor components in the rank 2 Zeeman Hamiltonian treatment. Arrays shown are for a single spin I=1/2 particle when the field axis is aligned along the +z axis.

We must very careful in using these single spin rank 2 tensors of this type because they contain both spatial and spin components. If we desire to express the Zeeman Hamiltonian relative to a particular set of axes we must insure that both the spatial tensor and the “spin” tensor are expressed in the proper coordinates. The spatial tensor alone cannot be rotated as it rotates only part of the spatial components[1]. It is improper to rotate this tensor in spin space because it also rotates spatial variables. Furthermore, note that **these rank 2 components are not the same as the rank 1 Zeeman tensor components**.

The 9 irreducible spherical components of a rank two spatial tensor, $A_{lm}^{(2)}$ , are related to its Cartesian components by the following formulas (See GAMMA Class Documentation on Spatial Tensor).

$$\begin{gathered}
A_{0,0} = \frac{-1}{\sqrt{3}}[A_{xx} + A_{yy} + A_{zz}] = \frac{-1}{\sqrt{3}}Tr\{A\} \\
A_{1,0} = \frac{-i}{\sqrt{2}}[A_{xy} - A_{yx}] \qquad A_{1,\pm1} = \frac{-1}{2}[A_{zx} - A_{xz} \pm i(A_{zy} - A_{yz})] \\
A_{2,0} = \sqrt{6}[3A_{zz} - (A_{xx} + A_{yy} + A_{zz})] = \sqrt{6}[3A_{zz} - Tr\{A\}] \\
A_{2,\pm1} = \mp\frac{1}{2}[A_{xz} + A_{zx} \pm i(A_{yz} + A_{zy})] \qquad A_{2,\pm2} = \frac{1}{2}[A_{xx} - A_{yy} \pm i(A_{xy} + A_{yx})]
\end{gathered} \tag{4-15}$$

Again the subscript *l* spans the rank as *l* = [0, 2], and the subscript *m* spans *+/- l, m* = [*-l, l*]. In this Zeeman treatment we then have the components $A_{l,m}^{Z2}$ as indicated in equation (4-33). Thus, the irreducible spherical tensor components can be obtained by substituting the Cartesian elements of **1** into equations (4-34). A general rank two Cartesian tensor can be rewritten in terms of a sum over tensors of ranks 0 through 2 as follows,

[1] See the discussion in Mehring

$$\begin{array}{ccccccccc} & & & & \text{Rank 0} & & \text{Rank 1} & & \text{Rank 2}\\ \hat{A}_i & = & \begin{bmatrix}A_{xx} & A_{xy} & A_{xz}\\ A_{yx} & A_{yy} & A_{yz}\\ A_{zx} & A_{zy} & A_{zz}\end{bmatrix}_i & = & A_{iso}(i)\begin{bmatrix}1&0&0\\0&1&0\\0&0&1\end{bmatrix} & + & \begin{bmatrix}0 & \alpha_{xy} & \alpha_{xz}\\ -\alpha_{xy} & 0 & \alpha_{yz}\\ -\alpha_{xz} & -\alpha_{yz} & 0\end{bmatrix}_i & + & \begin{bmatrix}\beta_{xx} & \beta_{xy} & \beta_{xz}\\ \beta_{yx} & \beta_{yy} & \beta_{yz}\\ \beta_{zx} & \beta_{zy} & \beta_{zz}\end{bmatrix}\end{array}$$

where

$$A_{iso} = \frac{1}{3}Tr\{\hat{A}\} \qquad \alpha_{xy} = \frac{1}{2}(A_{xy} - A_{yx}) \qquad \beta_{xy} = \frac{1}{2}(A_{xy} + A_{yx} - 2\delta_{xy}A_{iso})$$

The rank 0 part is isotropic (scalar), the rank 1 part is antisymmetric and traceless, and the rank 2 part traceless and symmetric. The Zeeman spatial tensor will be identical for each spin, $\hat{A}_i = \mathbf{1}$, so we have

$$A_{iso} = 1 \qquad \alpha_{xy} = 0 \qquad \beta_{xy} = 0$$

and the form remains the same

$$\begin{array}{ccccccccc} & & & & \text{Rank 0} & & \text{Rank 1} & & \text{Rank 2}\\ \hat{A}_i & = & \begin{bmatrix}1&0&0\\0&1&0\\0&0&1\end{bmatrix}_i & = & A_{iso}(i)\begin{bmatrix}1&1&1\\1&1&1\\1&1&1\end{bmatrix} & + & \begin{bmatrix}0&0&0\\0&0&0\\0&0&0\end{bmatrix}_i & + & \begin{bmatrix}0&0&0\\0&0&0\\0&0&0\end{bmatrix}\end{array}$$

Rank 2 spatial tensors are commonly specified in their principal axis system by the three components; the isotropic value $A_{iso}$, the anisotropy $\Delta A$, and the asymmetry $\eta$. These are generally given by

$$A_{iso} = \frac{1}{3}Tr\{A\}\ , \qquad \Delta A = A_{zz} - \frac{1}{2}(A_{xx} + A_{yy}) = \frac{3}{2}\delta_{zz} \qquad \eta = (\delta_{xx} - \delta_{yy})/\delta_{zz}$$

For the Zeeman spatial tensor we have

$$A_{iso} = 1, \qquad \Delta A = 0 \qquad \eta = 0 \tag{4-16}$$

and this is again the equation still in the original matrix form.

$$\hat{A}_i(PAS) = 1\begin{bmatrix}1&0&0\\0&1&0\\0&0&1\end{bmatrix} + \begin{bmatrix}0&0&0\\0&0&0\\0&0&0\end{bmatrix}_i + 0\begin{bmatrix}0&0&0\\0&0&0\\0&0&0\end{bmatrix}_i \tag{4-17}$$

The irreducible spherical elements of the Zeeman tensor, $A^Z_{2,\,m}$, in the principal axis system are, by placement of the identity matrix elements into (4-15),

## *Zeeman Rank 2 Irreducible Spherical Spatial Tensor Components (PAS)*

$$A^{Z2}_{0,0}(PAS) = -\sqrt{3}A_{iso} = -\sqrt{3}$$

$$A^{Z2}_{1,0}(PAS) = 0 \qquad A^{Z2}_{1,\pm 1}2(PAS) = 0$$

$$A^{Z2}_{2,0}(PAS) = 0 \qquad A^{Z2}_{2,\pm 1}(PAS) = 0 \qquad A^{Z2}_{2,\pm 2}(PAS) = 0$$

*Figure 4-12* : Principal axis components of the complete spatial tensor in the rank 2 treatment Zeeman Hamiltonian.

We can express the spatial tensor components $A^{Z2}_{l,m}$ relative to any arbitrary axis system (AAS) by a rotation from the principal axes to the new axes *via* the formula

$$A^{Z2}_{l,m}(i, AAS) = \sum_{m'}^{\pm l} D^{l}_{mm'}(\Omega)A^{Z2}_{l,m'}(i, PAS) \tag{4-18}$$

where $D^{l}_{mm'}$ are the rank $l$ Wigner rotation matrix elements and Ω the set of three Euler angles which relate the principal axes of the chemical Zeeman tensor to the arbitrary axes[1]. Since the only non-zero component of the Zeeman spatial tensor is of ranks l = 0, the equation describing it rotation will be very simple. Beginning with equation (4-14)

$$\boldsymbol{H}^{Z}_{i} = -h\gamma_i B_o \sum_{l=0}^{2} \sum_{m}^{\pm l} (-1)^m A^{Z2}_{l,m}(i) \bullet \boldsymbol{T}^{Z2}_{l,-m}(i)$$

we define a Zeeman interaction constant as

$$\xi^{Z}_{i} = h\gamma_i B_o \tag{4-19}$$

and expand the summation over the different ranks (only the rank 0 is non-zero).

$$\boldsymbol{H}^{Z}_{i} = -\xi^{Z}_{i} A^{Z2}_{0,0}(i)\boldsymbol{T}^{Z2}_{0,0}(i)$$

Rank 0 components are rotationally invariant, hence the Zeeman Hamiltonian is rotationally in-

[1] In this instance we should think to be careful in that express the elements $\boldsymbol{T}^{Z2}_{l,-m}$ in the same axis system as $A^{Z2}_{l,m}$. When $A^{Z2}$ is rotated in space, so must be $\boldsymbol{T}^{Z2}$. Essentially the field vector changes relative to any new coordinate system when constructing $\boldsymbol{T}^{Z2}$. However this point is inconsequential for the Zeeman treatment because, as we know, the Zeeman Hamiltonian will prove to be rotationally invariant.

variant as well, *i.e.* isotropic.

$$\boldsymbol{H}_i^Z(AAS) = -h\gamma_i B_o \sum_{l=0}^{2} \sum_{m}^{\pm l} (-1)^m A_{l,m}^{Z2}(i, AAS) \bullet \boldsymbol{T}_{l,-m}^{Z2}(i)$$

$$= -h\gamma_i B_o A_{0,0}^{Z2}(i, AAS) \bullet \boldsymbol{T}_{0,0}^{Z2}(i) = -h\gamma_i B_o \sum_{0}^{\pm 0} D_{00}^{0}(\Omega) A_{0,0}^{Z2}(i, PAS) \bullet \boldsymbol{T}_{0,0}^{Z2}(i)$$

$$= -h\gamma_i B_o A_{0,0}^{Z2}(i, PAS) \bullet \boldsymbol{T}_{0,0}^{Z2}(i) = -h\gamma_i B_o (-\sqrt{3})\left(\frac{-1}{\sqrt{3}} \boldsymbol{I}_{iz}\right)$$

And this is in agreement with the rank 1 treatment, equation (4-12).

$$\boldsymbol{H}_i^Z(AAS) = -h\gamma_i B_o \boldsymbol{I}_{iz} = -\Omega_{i,o} \boldsymbol{I}_{iz} \tag{4-20}$$

When working with an entire spin system one must sum over all spins with the tensors being in the same coordinate system, for our purposes the laboratory system. The Zeeman Hamiltonian for a spin system becomes the following.

$$\boldsymbol{H}^Z = \sum_{i}^{spins} \boldsymbol{H}_i^Z = \sum_{i}^{spins} -h\gamma_i B_o \boldsymbol{I}_{iz} = \sum_{i}^{spins} -\Omega_{i,o} \boldsymbol{I}_{iz} \tag{4-21}$$

The following figures summarize the rank 2 treatment of the Zeeman Hamiltonian.

## Zeeman Rank 2 Treatment Hamiltonian Summary

$$\boldsymbol{H}^Z(AAS) = \sum_i^{spins} \boldsymbol{H}_i^Z(AAS) = \sum_i^{spins} \xi_i^Z \sum_{l=0}^{2} \sum_m^{\pm l} (-1)^m A^Z_{l-m}(i, AAS) \bullet \boldsymbol{T}^Z_{lm}(i)$$

$$\boldsymbol{H}_i^Z(AAS) = \xi_i^Z \sum_{l=0}^{2} \sum_m^{\pm 2} (-1)^m A^Z_{l-m}(i, AAS) \boldsymbol{T}^Z_{lm}(i, AAS)$$

$$\xi_i^Z = h\gamma_i B_o$$

$$A^{Z2}_{l,m}(i, AAS) = \sum_{m'}^{\pm l} D^l_{mm'}(\varphi, \theta, \chi) A^{Z2}_{l,m'}(i, PAS)$$

$$A^{Z2}_{0,0}(i, PAS) = -\sqrt{3}$$

$$A^{Z2}_{1,0}(i, PAS) = 0 \qquad A^{Z2}_{1,\pm1}(i, PAS) = 0$$

$$A^{Z2}_{2,0}(i, PAS) = 0 \qquad A^{Z2}_{2,\pm1}(i, PAS) = 0 \qquad A^{Z2}_{2,\pm2}(i, PAS) = 0$$

$$T^Z_{0,0}(i, AAS) = \frac{-1}{\sqrt{3}}\left[\boldsymbol{I}_{iz}B_z + \frac{1}{2}(\boldsymbol{I}_{i+}B_- + \boldsymbol{I}_{i-}B_+)\right] = \frac{-1}{\sqrt{3}}\vec{\boldsymbol{I}}_i \bullet \vec{\boldsymbol{B}}_n$$

$$T^Z_{1,0}(i, AAS) = \frac{-1}{2\sqrt{2}}[\boldsymbol{I}_{i+}B_- - \boldsymbol{I}_{i-}B_+] \qquad T^Z_{1,\pm1}(i, AAS) = \frac{-1}{2}[\boldsymbol{I}_{i\pm}\ B_z - \boldsymbol{I}_{iz}B_\pm\ ]$$

$$T^Z_{2,0}(i, AAS) = \frac{1}{\sqrt{6}}[3\boldsymbol{I}_{iz}B_z - (\vec{\boldsymbol{I}}_i \bullet \vec{\boldsymbol{B}}_n)]$$

$$T^Z_{2,\pm1}(i, AAS) = \mp\frac{1}{2}[\boldsymbol{I}_{i\pm}\ B_z + \boldsymbol{I}_{iz}B_\pm\ ] \qquad T^Z_{2,\pm2}(i, AAS) = \frac{1}{2}[\boldsymbol{I}_{i\pm}\ B_\pm\ ]$$

*Figure 4-13* : Principal axis components of the complete spatial tensor in the rank 2 treatment Zeeman Hamiltonian.

## 8.7.4 The Chemical Shielding Hamiltonian

A chemical shift is the observed effect from the electron cloud surrounding a nucleus responding to an applied magnetic field. The spin itself experiences not only the applied field but also a field from the perturbed electron cloud, the latter field generally opposing the applied field or "shielding" the nucleus. We can write this latter "induced" field in terms of the applied field, $\vec{\boldsymbol{B}}_o$, as

$$\vec{\boldsymbol{B}}_{induced} = -\hat{\sigma} \bullet \vec{\boldsymbol{B}}_o$$

where $\hat{\sigma}$ is the chemical shielding tensor, a 3x3 array in Cartesian space, and the $\vec{\boldsymbol{B}}$'s vectors in Cartesian space. In matrix form this is simply[1]

$$\begin{bmatrix} B_{ind,\,x} \\ B_{ind,\,y} \\ B_{ind,\,z} \end{bmatrix} = -\begin{bmatrix} \sigma_{xx} & \sigma_{xy} & \sigma_{xz} \\ \sigma_{yx} & \sigma_{yy} & \sigma_{yz} \\ \sigma_{zx} & \sigma_{zy} & \sigma_{zz} \end{bmatrix}_i \bullet \begin{bmatrix} B_{0x} \\ B_{0y} \\ B_{0z} \end{bmatrix},$$

the induced field depends on the applied field strength, the applied field orientation, and the surrounding electron cloud. Note that $\vec{\boldsymbol{B}}_{induced}$ will not necessarily be co-linear with the applied field. Of course, every nuclear spin will have its own associated chemical shielding tensor. The classical interaction energy between this induced field and a nuclear spin is

$$E_i^{CS} = -\vec{\mu}_i \bullet \vec{\boldsymbol{B}}_{induced} = \vec{\mu}_i \bullet \hat{\sigma}_i \bullet \vec{\boldsymbol{B}}_o$$

where $\vec{\mu}$ is the magnetic moment, $i$ the spin index, $E$ the energy, and subscript $CS$ used to denote chemical shielding. The associated Hamiltonian is obtained from substitution of $h\gamma\vec{\boldsymbol{I}}_i$ for $\vec{\mu}_i$.

$$\boldsymbol{H}_i^{CS} = h\gamma_i \vec{\boldsymbol{I}}_i \bullet \hat{\sigma}_i \bullet \vec{\boldsymbol{B}}_o \qquad (4\text{-}22)$$

In matrix form this equation looks like

$$\boldsymbol{H}_i^{CS} = h\gamma_i \begin{bmatrix} \boldsymbol{I}_{ix} & \boldsymbol{I}_{iy} & \boldsymbol{I}_{iz} \end{bmatrix} \bullet \begin{bmatrix} \sigma_{xx} & \sigma_{xy} & \sigma_{xz} \\ \sigma_{yx} & \sigma_{yy} & \sigma_{yz} \\ \sigma_{zx} & \sigma_{zy} & \sigma_{zz} \end{bmatrix}_i \bullet \begin{bmatrix} B_{0x} \\ B_{0y} \\ B_{0z} \end{bmatrix}. \qquad (4\text{-}23)$$

Taking the magnitude of the applied field out, equation (4-22) is simply

$$\boldsymbol{H}_i^{CS} = h\gamma_i B_o \sum_u^{axes} \sum_v^{axes} \langle 1|\vec{\boldsymbol{I}}_i|u\rangle\langle u|\hat{\sigma}_i|v\rangle\langle v|\vec{\boldsymbol{B}}_n|1\rangle \qquad (4\text{-}24)$$

[1] Note that the effect of the chemical shielding is to alter the field which the spin experiences. This is clearly seen from the product $\hat{\sigma}_i \bullet \vec{\boldsymbol{B}}_o$ which produces an effective field vector for the spin.

with $u, v \in \{x, y, z\}$ and $\vec{\boldsymbol{B}}_n$ a normalized magnetic field vector in the direction of the applied field.

## Rank 1 Chemical Shielding Hamiltonian Treatment

Performing the latter matrix-vector multiplication of equation (4-23) produces

$$\boldsymbol{H}_i^{CS} = h\gamma_i B_o \begin{bmatrix} \boldsymbol{I}_{ix} & \boldsymbol{I}_{iy} & \boldsymbol{I}_{iz} \end{bmatrix} \bullet \begin{bmatrix} \sigma_{xx}B_{nx} + \sigma_{xy}B_{ny} + \sigma_{xz}B_{nz} \\ \sigma_{yx}B_{nx} + \sigma_{yy}B_{ny} + \sigma_{yz}B_{nz} \\ \sigma_{zx}B_{nx} + \sigma_{zy}B_{ny} + \sigma_{zz}B_{nz} \end{bmatrix}_i \tag{4-25}$$

Letting[1]

$$\boldsymbol{T}_i^{CS1} = \begin{bmatrix} \boldsymbol{I}_{ix} & \boldsymbol{I}_{iy} & \boldsymbol{I}_{iz} \end{bmatrix} \qquad \text{and} \qquad \boldsymbol{A}_i^{CS1} = \begin{bmatrix} \sigma_{xx}B_{nx} + \sigma_{xy}B_{ny} + \sigma_{xz}B_{nz} \\ \sigma_{yx}B_{nx} + \sigma_{yy}B_{ny} + \sigma_{yz}B_{nz} \\ \sigma_{zx}B_{nx} + \sigma_{zy}B_{ny} + \sigma_{zz}B_{nz} \end{bmatrix}_i \tag{4-26}$$

the shielding Hamiltonian be placed into the context of a scalar product of two rank 1 Cartesian tensors (vectors).

$$\boldsymbol{H}_i^{CS} = h\gamma_i B_o \boldsymbol{A}_i^{CS1} \bullet \boldsymbol{T}_i^{CS1}$$

or (4-27)

$$\boldsymbol{H}_i^{CS} = h\gamma_i B_o \sum_u^{axes} \langle 1|\boldsymbol{A}_i^{CS1}|u\rangle\langle u|\boldsymbol{T}_i^{CS1}|1\rangle$$

Rewriting equation (4-27) in terms of 4 irreducible spherical rank 1 tensor components produces

$$\boldsymbol{H}_i^{CS} = h\gamma_i B_o \sum_{l=0}^{1} \sum_{m}^{\pm l} (-1)^m \boldsymbol{A}_{l,-m}^{CS1}(i) \bullet \boldsymbol{T}_{lm}^{CS1}(i) \quad . \tag{4-28}$$

We obtain the 4 irreducible spherical components of the CS rank 1 spin tensor directly from the Cartesian components, $\langle v|\hat{\boldsymbol{T}}_i^{CS1}|u\rangle$ , as indicated in GAMMA Class Documentation on Spin Tensor. These are

$$\boldsymbol{T}_{l,m}^{CS1}(i) \quad ,$$

where $CS1$ signifies the rank 1 treatment of the chemical shielding interaction and $i$ is the spin index. The tensor index $l$ spans the rank: $l \in [0, 1]$ while the tensor index $m$ spans $l$: $m \in [-l, l]$ The four formulas for these quantities a listed in the following figure.

[1] We have switched from the superscript $CS$ to use of $CS1$ to indicate the rank 1 treatment. This will distinguish it from the rank 2 treatment presented later in this section.

## *Rank 1 Irreducible Spherical Spin Tensor Components*

$$T_{0,0}(i) = 0 \qquad T_{1,0}(i) = I_{iz} \qquad T_{1,\pm 1}(i) = \frac{\mp 1}{\sqrt{2}} I_{i\pm}$$

For a single spin 1/2 particle the chemical shielding spin tensor components are, in the Hilbert space of the spin itself, given in the following figure (the spin index is implicit here)

## *Zeeman Hamiltonian Summary*

### *Chemical Shielding Rank 1 Irreducible Spherical Spin Tensor Components*
### *Matrix Representations in 1-spin (I=1/2) Hilbert Space*

$$T^{(1)}_{0,0} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \qquad T^{(1)}_{1,0} = \frac{1}{2}\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \qquad T^{(1)}_{1,-1} = \frac{1}{\sqrt{2}}\begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix} \qquad T^{(1)}_{1,1} = \frac{-1}{\sqrt{2}}\begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}$$

The 4 irreducible spherical components of the chemical shielding spatial tensor (rank 1) are formally specified with the nomenclature

$$A^{CS1}_{l,m}(i),$$

where again the subscript *l* spans the rank as *l* = [0, 1], and the subscript *m* spans +/- *l*, *m* = [-*l*, *l*]. The four formulas which produce these quantities are

## *Rank 1 Chemical Shielding Irreducible Spherical Spatial Tensor Components*

$$A^{CS1}_{0,0} = 0$$

$$A^{CS1}_{1,0} = A^{CS1}_{zz} = \sigma_{zx}B_{nx} + \sigma_{zy}B_{ny} + \sigma_{zz}B_{nz}$$

$$A^{CS1}_{1,\pm 1} = \mp A^{CS1}_{\pm} = (\sigma_{xx}B_{nx} + \sigma_{xy}B_{ny} + \sigma_{xz}B_{nz}) \pm i(\sigma_{yx}B_{nx} + \sigma_{yy}B_{ny} + \sigma_{yz}B_{nz})$$

If we express the spatial tensor (the field vector) in the laboratory axes, and the normalized vector is given by $\vec{\boldsymbol{B}}_n = \vec{k}$, we have

## *Rank 1 Chemical Shielding Irreducible Spherical Spatial Tensor Components*

### *Rank 1 Chemical Shielding Irreducible Spherical Spatial Tensor Components*
### *Aligned Along the z-axis of the Magnetic Field Vector*

$$A^{CS1}_{0,0}(i, LAB) = 0$$

$$A^{CS1}_{1,0}(i, LAB) = \sigma_{zz}(i, LAB) \qquad A^{CS1}_{1,\pm 1}(i, LAB) = \sigma_{xz}(i, LAB) \pm i\sigma_{yz}(i, LAB)$$

It is common to specify rank 2 spatial tensors in their principal axis system by the three components; the isotropic value $A_{iso}$, the anisotropy $\Delta A$, and the asymmetry $\eta$. These are generally given by

$$A_{iso} = \frac{1}{3}Tr\{A\} \ , \qquad \Delta A = A_{zz} - \frac{1}{2}(A_{xx} + A_{yy}) = \frac{3}{2}\delta_{zz} \qquad \eta = (\delta_{xx} - \delta_{yy})/\delta_{zz}$$

Thus, the shielding tensor (Cartesian) is given in its principal axis system by

$$\hat{\sigma}_i(PAS) = \sigma_{iso}(i)\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} + \begin{bmatrix} 0 & \alpha_{xy} & \alpha_{xz} \\ -\alpha_{xy} & 0 & \alpha_{yz} \\ -\alpha_{xz} & -\alpha_{yz} & 0 \end{bmatrix}_i + \delta_{zz}\begin{bmatrix} -\frac{1}{2}(1-\eta) & 0 & 0 \\ 0 & -\frac{1}{2}(1+\eta) & 0 \\ 0 & 0 & 1 \end{bmatrix}_i$$

Using this form we can attain the components of the rank 1 $A^{CS1}_{l,\,m}$ in the event that the rank 1 component (the asymmetric) term is zero or can be neglected.

### Rank 1 Chemical Shielding Irreducible Spherical Spatial Tensor Components

### Rank 1 Chemical Shielding Irreducible Spherical Spatial Tensor Components Principal Axis System of the Symmetric Shielding Tensor

$$A^{CS1}_{0,\,0}(PAS_\sigma) = 0$$

$$A^{CS1}_{1,\,0}(PAS_\sigma) = A^{CS1}_{zz}(PAS_\sigma) = [\sigma_{iso} + \delta_{zz}]B_{nz}$$

$$A^{CS1}_{1,\,\pm1}(PAS_\sigma) = \mp A^{CS1}_{\pm}(PAS_\sigma) = \left[\sigma_{iso} - \frac{\delta_{zz}}{2}(1-\eta)\right]B_{nx} \pm i\left[\sigma_{iso} - \frac{\delta_{zz}}{2}(1+\eta)\right]B_{ny}$$

Defining the chemical shielding interaction constant as[1]

$$\xi^{CS}_i = h\gamma_i B_o \ , \qquad (4\text{-}29)$$

The chemical shielding Hamiltonian is given by

$$\boldsymbol{H}^{CS}_i = \xi^{CS}_i \sum_m^{\pm1} (-1)^m \boldsymbol{A}^{CS1}_{1,\,-m}(i) \bullet \boldsymbol{T}^{CS1}_{1m}(i) \qquad (4\text{-}30)$$

and by summing over all the spin in a spin system we attain

[1] This will be the same as the chemical shielding rank 2 treatment’s interaction constant.

$$\boldsymbol{H}^{CS} = \sum_{i}^{spins} \xi_i^{CS} \sum_{m}^{\pm 1} (-1)^m \boldsymbol{A}_{1,-m}^{CS1}(i) \bullet \boldsymbol{T}_{1m}^{CS1}(i) \qquad (4\text{-}31)$$

We now regroup all the applicable equations for dealing with the chemical shielding Hamiltonian based on the rank 1 treatment.

## *The Rank 1 Chemical Shielding Hamiltonian Summary*

Arbitrary Axis System

$$\boldsymbol{H}^{CS}(AAS) = \sum_{i}^{spins} \boldsymbol{H}_i^{CS}(AAS) = \sum_{i}^{spins} \xi_i^{CS} \sum_{l=0}^{1} \sum_{m}^{\pm l} (-1)^m A_{l-m}^{CS1}(i, AAS) \boldsymbol{T}_{lm}^{CS1}(i, AAS)$$

$$\xi_i^{CS} = h\gamma_i B_o$$

$$A_{0,0}^{CS1}(AAS) = 0$$

$$A_{1,0}^{CS1}(AAS) = \sigma_{zx}B_{nx} + \sigma_{zy}B_{ny} + \sigma_{zz}B_{nz}$$

$$A_{1,\pm1}^{CS1}(AAS) = (\sigma_{xx}B_{nx} + \sigma_{xy}B_{ny} + \sigma_{xz}B_{nz}) \pm i(\sigma_{yx}B_{nx} + \sigma_{yy}B_{ny} + \sigma_{yz}B_{nz})$$

$$T_{0,0}^{CS1}(i, AAS) = 0 \qquad T_{1,0}^{CS1}(i, AAS) = I_{iz} \qquad T_{1,\pm1}^{CS1}(i, AAS) = \frac{\mp 1}{\sqrt{2}} I_{i\pm}$$

Laboratory Frame

$$A_{0,0}^{CS1}(LAB) = 0 \qquad A_{1,0}^{CS1}(LAB) = \sigma_{zz} \qquad A_{1,\pm1}^{CS1}(LAB) = \sigma_{xz} \pm i\sigma_{yz}$$

Shielding Tensor Principal Axes

$$A_{0,0}^{CS1}(PAS) = 0$$

$$A_{1,0}^{CS1}(PAS) = \sigma_{zz}B_{nz} = [\sigma_{iso} + \delta_{zz}]B_{nz}$$

$$A_{1,\pm1}^{CS1}(PAS) = \sigma_{xx}B_{nx} \pm i\sigma_{yy}B_{ny} = \left[\sigma_{iso} - \frac{\delta_{zz}}{2}(1-\eta)\right]B_{nx} \pm i\left[\sigma_{iso} - \frac{\delta_{zz}}{2}(1+\eta)\right]B_{ny}$$

The rank 1 treatment presented in this section is often used in treatment of spin systems in the solid state. Normally, the shielding tensor, $\hat{\sigma}_i$, for a particular spin is known in its principal axis system whereas the applied static field is known in the laboratory frame (a vector pointing along the positive z-axis). To express $\boldsymbol{H}_i^{CS}$ in the principal axis system of $\hat{\sigma}_i$ one rotates the normalized field vector $\vec{B}_n$ into the principal axis system of $\hat{\sigma}_i$ using three Euler angles to specify the relative orientations. Once $\boldsymbol{H}_i^{CS}(PAS_\sigma)$ is obtained, one may readily obtain $\boldsymbol{H}_i^{CS}(AAS)$ by rotating $A^{CS1}(AAS)$ by another set of Euler angles. For example, a powder average can be attained this way by averaging over all possible orientations of $A^{CS1}$.

## Rank 2 Chemical Shielding Hamiltonian Treatment

Equation (4-23) can also be rearranged to produce an equation involving two rank 2 tensors by taking the dyadic product of the vectors $\boldsymbol{I}_i$ and $\boldsymbol{B}_n$.

$$\boldsymbol{H}_i^{CS} = h\gamma_i B_o \sum_u^{axes} \sum_v^{axes} \langle u|\sigma_i|v\rangle\langle v|\boldsymbol{B}_n|1\rangle\langle 1|\boldsymbol{I}_i|u\rangle = h\gamma_i B_o \sum_u^{axes} \sum_v^{axes} \langle u|\sigma_i|v\rangle\langle v|\boldsymbol{B}_n\boldsymbol{I}_i|u\rangle$$

The dyadic product to produce $\boldsymbol{B}_n\boldsymbol{I}_i$ is explicitly done *via*

$$\begin{bmatrix} B_{nx} \\ B_{ny} \\ B_{nz} \end{bmatrix} \bullet \begin{bmatrix} \boldsymbol{I}_{ix} & \boldsymbol{I}_{iy} & \boldsymbol{I}_{iz} \end{bmatrix} = \begin{bmatrix} B_{nx}\boldsymbol{I}_{xi} & B_{nx}\boldsymbol{I}_{yi} & B_{nx}\boldsymbol{I}_{zi} \\ B_{ny}\boldsymbol{I}_{xi} & B_{ny}\boldsymbol{I}_{yi} & B_{ny}\boldsymbol{I}_{zi} \\ B_{nz}\boldsymbol{I}_{xi} & B_{nz}\boldsymbol{I}_{yi} & B_{nz}\boldsymbol{I}_{zi} \end{bmatrix}.$$

The chemical shielding Hamiltonian can thus be formulated as a scalar product of two rank 2 tensors. Letting $\hat{\boldsymbol{T}}_i = \vec{\boldsymbol{B}}_n\vec{\boldsymbol{I}}_i$, we have

$$\boldsymbol{H}_i^{CS} = h\gamma_i B_o \hat{\sigma}_i \bullet \hat{\boldsymbol{T}}_i \ .$$

or equivalently (4-32)

$$\boldsymbol{H}_i^{CS} = h\gamma_i B_o \sum_u^{axes} \sum_v^{axes} \langle u|\sigma_i|v\rangle\langle v|\boldsymbol{T}_i|u\rangle$$

Rewriting equation (4-32) in terms of irreducible spherical components rather than the current Cartesian components

$$\boldsymbol{H}_i^{CS} = h\gamma_i B_o \sum_{l=0}^{2} \sum_m^{\pm l} (-1)^m A_{l,-m}^{CS2}(i) \bullet \boldsymbol{T}_{l,m}^{CS2}(i) \qquad (4\text{-}33)$$

We can obtain the 9 irreducible spherical components of the CS rank 2 "spin" tensor[1] directly from the Cartesian components, $\langle v|\hat{\boldsymbol{T}}_i|u\rangle$ , as indicated in GAMMA Class Documentation on Spin Tensor. These are

$$\boldsymbol{T}_{l,m}^{CS2}(i) \ ,$$

[1] Due to the nature of the CS interaction, the rank 2 tensor treatment produces a "spin" tensor $T_{l,m}^{CS2}(i)$ which contains spatial components, namely the magnetic field vector. As a result, care must be used when performing spatial rotations on shielding tensors. Any spatial rotations must involve rotations of both $\sigma$ and $\boldsymbol{T}$

where $CS$ signifies the chemical shielding interaction, and $i$ is the spin index. The tensor index $l$ spans the rank: $l \in [0, 2]$ while the tensor index $m$ spans $l$: $m \in [-l, l]$ The nine formulas for these quantities a listed in the following figure where the field components are those of the normalized field vector $\vec{\boldsymbol{B}}_n$.[1]

### *Shielding Rank 2 Irreducible Spherical Spin-Space Tensor Components*

$$T_{0,0}(i) = \frac{-1}{\sqrt{3}}\left[\boldsymbol{I}_{iz}B_z + \frac{1}{2}(\boldsymbol{I}_{i+}B_- + \boldsymbol{I}_{i-}B_+)\right] = \frac{-1}{\sqrt{3}}\vec{\boldsymbol{I}}_i \bullet \vec{\boldsymbol{B}}_n$$

$$T_{1,0}(i) = \frac{-1}{2\sqrt{2}}[\boldsymbol{I}_{i+}B_- - \boldsymbol{I}_{i-}B_+] \qquad T_{1,\pm 1}(i) = \frac{-1}{2}[\boldsymbol{I}_{i\pm}B_z - \boldsymbol{I}_{iz}B_\pm]$$

$$T_{2,0}(i) = \frac{1}{\sqrt{6}}[3\boldsymbol{I}_{iz}B_z - (\vec{\boldsymbol{I}}_i \bullet \vec{\boldsymbol{B}}_n)]$$

$$T_{2,\pm 1}(i) = \mp\frac{1}{2}[\boldsymbol{I}_{i\pm}B_z + \boldsymbol{I}_{iz}B_\pm] \qquad T_{2,\pm 2}(i) = \frac{1}{2}[\boldsymbol{I}_{i\pm}B_\pm]$$

For a spin 1/2 particle and $\vec{B}_0 = B_0\vec{B}_n$ s, the matrix form of these tensor components are shown in the following figure in the single spin Hilbert space. The spin index has been omitted, the field components are those of the normalized vector $\vec{B}_n$.

### *The Rank 1 Chemical Shielding Hamiltonian Summary*

**Rank 2 Chemical Shielding Irreducible Spherical Spin-Space Tensor Component**
**Matrix Representations in 1-spin (I=1/2) Hilbert Space**

$$T^{(2)}_{0,0} = \frac{-1}{2\sqrt{3}}\begin{bmatrix} B_z & B_- \\ B_+ & -B_z \end{bmatrix} \quad T^{(2)}_{1,0} = \frac{-1}{2\sqrt{2}}\begin{bmatrix} 0 & B_- \\ -B_+ & 0 \end{bmatrix} \quad T^{(2)}_{1,-1} = \frac{-1}{2}\begin{bmatrix} -B_-/2 & 0 \\ B_z & B_-/2 \end{bmatrix} \quad T^{(2)}_{1,1} = \frac{-1}{2}\begin{bmatrix} -B_+/2 & B_z \\ 0 & B_+/2 \end{bmatrix}$$

$$T^{(2)}_{2,0} = \frac{1}{2\sqrt{6}}\begin{bmatrix} 2B_z & -B_- \\ -B_+ & -2B_z \end{bmatrix} \quad T^{(2)}_{2,-1} = \frac{1}{2}\begin{bmatrix} B_-/2 & B_z \\ 0 & -B_-/2 \end{bmatrix} \quad T^{(2)}_{2,1} = \frac{-1}{2}\begin{bmatrix} B_+/2 & 0 \\ B_z & -B_+/2 \end{bmatrix} \quad T^{(2)}_{2,-2} = \frac{1}{2}\begin{bmatrix} 0 & 0 \\ B_- & 0 \end{bmatrix} \quad T^{(2)}_{2,2} = \frac{1}{2}\begin{bmatrix} 0 & B_+ \\ 0 & 0 \end{bmatrix}$$

The raising an lowering components of the field vector are defined in the standard fashion, namely $\vec{B}_\pm = B_x \pm iB_y$. The simplest situation occurs when magnetic field points along the positive z-

[1] For these formulae, it is important to note that it is the second component in the composite spin/space tensor which is set to the normalized magnetic field vector $\vec{B}_n$, 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. Our field vector has be set to point along the positive z-axis in the laboratory frame.

axis, $\vec{\boldsymbol{B}}_n = \hat{\boldsymbol{k}}$, *i.e.* these spin-space tensors are written in the laboratory frame. Then, the (normalized) field vector simplifies, $\vec{B}_z = 1$ and $B_x = B_y = \vec{B}_{\pm} = 0$. The applicable equations for the shielding space-spin tensors are then as follows.

### The Rank 1 Chemical Shielding Hamiltonian Summary

**ank 2 Chemical Shielding Irreducible Spherical Spin-Space Tensor Component**
**Aligned Along the z-axis of the Magnetic Field Vector**

$$T^{CS2}_{0,0}(ij) = \frac{-1}{\sqrt{3}}\boldsymbol{I}_{iz}$$

$$T^{CS2}_{1,0}(ij) = 0 \qquad\qquad T^{CS2}_{1,\pm 1}(ij) = \frac{-1}{2}\boldsymbol{I}_{i\pm}$$

$$T^{CS2}_{2,0}(ij) = \frac{2}{\sqrt{6}}\boldsymbol{I}_{iz}$$

$$T^{CS2}_{2,\pm 1}(ij) = \mp\frac{1}{2}\boldsymbol{I}_{i\pm} \qquad\qquad T^{CS2}_{2,\pm 2}(ij) = 0$$

For a spin 1/2 particle and $\vec{B}_0 = B_0\vec{B}_n$ along the positive z-axis, the matrix form of these tensor components are shown in the following figure[1] (in the single spin Hilbert space).

### The Rank 1 Chemical Shielding Hamiltonian Summary

**ank 2 Chemical Shielding Irreducible Spherical Spin-Space Tensor Component**
**Matrix Representations in 1-spin (I=1/2) Hilbert Space**
**Aligned Along the z-axis of the Magnetic Field Vector**

$$T^{CS2}_{0,0} = \frac{-1}{2\sqrt{3}}\begin{bmatrix}1 & 0\\ 0 & -1\end{bmatrix} \qquad T^{CS2}_{1,0} = \begin{bmatrix}0 & 0\\ 0 & 0\end{bmatrix} \qquad T^{CS2}_{1,-1} = \frac{-1}{2}\begin{bmatrix}0 & 0\\ 1 & 0\end{bmatrix} \qquad T^{CS2}_{1,1} = \frac{-1}{2}\begin{bmatrix}0 & 1\\ 0 & 0\end{bmatrix}$$

$$T^{CS2}_{2,0} = \frac{1}{\sqrt{6}}\begin{bmatrix}1 & 0\\ 0 & -1\end{bmatrix} \qquad T^{CS2}_{2,1} = \frac{-1}{2}\begin{bmatrix}0 & 1\\ 0 & 0\end{bmatrix} \qquad T^{CS2}_{2,-1} = \frac{1}{2}\begin{bmatrix}0 & 0\\ 1 & 0\end{bmatrix} \qquad T^{CS2}_{2,-2} = \begin{bmatrix}0 & 0\\ 0 & 0\end{bmatrix} \qquad T^{CS2}_{2,2} = \begin{bmatrix}0 & 0\\ 0 & 0\end{bmatrix}$$

We must very careful in using these single spin rank 2 shielding tensors of this type because they contain both spatial and spin components. If we desire to express the shielding Hamiltonian relative to a particular set of axes we must insure that both the spatial tensor and the "spin" tensor are expressed in the proper coordinates. The spatial tensor alone cannot be rotated as it rotates only part of the spatial components[2]. It is improper to rotate this tensor in spin space because it also ro-

[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

tates spatial variables.Furthermore, note that **these rank 2 components are not the same as the single spin rank 1 chemical shielding tensor components**.

The 9 irreducible spherical components of a rank two spatial tensor, $A^{(2)}_{lm}$ , are related to its Cartesian components by the following formulas (See GAMMA Class Documentation on Spatial Tensor).

$$\begin{gathered}
A_{0,\,0} \;=\; \frac{-1}{\sqrt{3}}[A_{xx}+A_{yy}+A_{zz}] \;=\; \frac{-1}{\sqrt{3}}Tr\{A\} \\
A_{1,\,0} \;=\; \frac{-i}{\sqrt{2}}[A_{xy}-A_{yx}] \qquad A_{1,\,\pm1} \;=\; \frac{-1}{2}[A_{zx}-A_{xz}\pm i(A_{zy}-A_{yz})] \\
A_{2,\,0} \;=\; \sqrt{6}[3A_{zz}-(A_{xx}+A_{yy}+A_{zz})] \;=\; \sqrt{6}[3A_{zz}-Tr\{A\}] \\
A_{2,\,\pm1} \;=\; \mp\frac{1}{2}[A_{xz}+A_{zx}\pm i(A_{yz}+A_{zy})] \qquad A_{2,\,\pm2} \;=\; \frac{1}{2}[A_{xx}-A_{yy}\pm i(A_{xy}+A_{yx})]
\end{gathered} \tag{4-34}$$

Again the subscript *l* spans the rank as *l* = [0, 2], and the subscript *m* spans *+/- l, m* = [*-l*, *l*]. In this chemical shielding treatment we then have the components $A^{CS2}_{l,\,m}$ as indicated in equation (4-33). Thus, the irreducible spherical tensor components can be obtained by substituting the Cartesian elements of $\hat{\sigma}_i$ into equations (4-34). A general rank two Cartesian tensor can be rewritten in terms of a sum over tensors of ranks 0 through 2 as follows,

Rank 0 Rank 1 Rank 2

$$\grave{\text{ɔ}}_i \;=\; \begin{bmatrix} \sigma_{xx} & \sigma_{xy} & \sigma_{xz} \\ \sigma_{yx} & \sigma_{yy} & \sigma_{yz} \\ \sigma_{zx} & \sigma_{zy} & \sigma_{zz} \end{bmatrix}_i \;=\; \sigma_{iso}(i)\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} + \begin{bmatrix} 0 & \alpha_{xy} & \alpha_{xz} \\ -\alpha_{xy} & 0 & \alpha_{yz} \\ -\alpha_{xz} & -\alpha_{yz} & 0 \end{bmatrix}_i + \begin{bmatrix} \delta_{xx} & \delta_{xy} & \delta_{xz} \\ \delta_{yx} & \delta_{yy} & \delta_{yz} \\ \delta_{zx} & \delta_{zy} & \delta_{zz} \end{bmatrix}$$

where

$$\sigma_{iso} \;=\; \frac{1}{3}Tr\{\hat{\sigma}\} \qquad \alpha_{xy} \;=\; \frac{1}{2}(\sigma_{xy}-\sigma_{yx}) \qquad \delta_{xy} \;=\; \frac{1}{2}(\sigma_{xy}+\sigma_{yx}-2\sigma_{iso})$$

The rank 0 part is isotropic (scalar), the rank 1 part is antisymmetric and traceless, and the rank 2 part traceless and symmetric. As with the dipolar spatial tensor, the chemical shielding spatial tensor can be specified most readily in its principal axis system. Unlike dipolar spatial tensors which all appear the same in their respective principal axes, the shielding tensors will normally be different for each spin. The shielding tensor values are experimentally determined in the tensor principal axes, and then less values are necessary to specify the tensor because many more tensor elements are zero.

$$\hat{\sigma}_i(PAS) = \sigma_{iso}(i)\begin{bmatrix}1&0&0\\0&1&0\\0&0&1\end{bmatrix} + \begin{bmatrix}0&\alpha_{xy}&\alpha_{xz}\\-\alpha_{xy}&0&\alpha_{yz}\\-\alpha_{xz}&-\alpha_{yz}&0\end{bmatrix}_i + \begin{bmatrix}\delta_{xx}&0&0\\0&\delta_{yy}&0\\0&0&\delta_{zz}\end{bmatrix}_i$$

Rank 2 spatial tensors are commonly specified in their principal axis system by the three components; the isotropic value $A_{iso}$, the anisotropy $\Delta A$, and the asymmetry $\eta$. These are generally given by

$$A_{iso} = \frac{1}{3}Tr\{A\} \quad , \qquad \Delta A = A_{zz} - \frac{1}{2}(A_{xx} + A_{yy}) = \frac{3}{2}\delta_{zz} \qquad \eta = (\delta_{xx} - \delta_{yy})/\delta_{zz}$$

A set of Euler angles $\{\alpha, \beta, \gamma\}$ is normally also given to relate the spatial tensor principle axes to another coordinate system. For the shielding spatial tensor we have

$$\hat{\sigma}_i(PAS) = \sigma_{iso}(i)\begin{bmatrix}1&0&0\\0&1&0\\0&0&1\end{bmatrix} + \begin{bmatrix}0&\alpha_{xy}&\alpha_{xz}\\-\alpha_{xy}&0&\alpha_{yz}\\-\alpha_{xz}&-\alpha_{yz}&0\end{bmatrix}_i + \delta_{zz}\begin{bmatrix}-\frac{1}{2}(1-\eta)&0&0\\0&-\frac{1}{2}(1+\eta)&0\\0&0&1\end{bmatrix}_i \qquad (4\text{-}35)$$

The irreducible spherical elements of the shielding tensor, $A^{CS}_{2,m}$, in the principal axis system are, by placement of (4-35) into (4-34),

### The Rank 1 Chemical Shielding Hamiltonian Summary

#### Chemical Shielding Rank 2 Irreducible Spherical Spatial Tensor Component
#### Principal Axis System (PAS)

$$A^{CS2}_{0,0}(PAS) = -\sqrt{3}\sigma_{iso}$$

$$A^{CS2}_{1,0}(PAS) = -\frac{i}{\sqrt{2}}[\sigma_{xy} - \sigma_{yx}] \qquad A^{CS}_{1,\pm1}2(PAS) = -\frac{1}{2}[(\sigma_{zx} - \sigma_{xz}) \pm i(\sigma_{zy} - \sigma_{yz})]$$

$$A^{CS2}_{2,0}(PAS) = \sqrt{3/2}\delta_{zz} \qquad A^{CS2}_{2,\pm1}(PAS) = 0 \qquad A^{CS2}_{2,\pm2}(PAS) = \frac{1}{2}\delta_{zz}\eta$$

We can express the spatial tensor components $A^{CS}_{l,m}$ relative to any arbitrary axis system (AAS) by a rotation from the principal axes to the new axes *via* the formula

$$A_{l,m}^{CS2}(i, AAS) = \sum_{m'}^{\pm l} D_{mm'}^{l}(\Omega) A_{l,m'}^{CS2}(i, PAS) \tag{4-36}$$

where $D_{mm'}^{l}$ are the rank $l$ Wigner rotation matrix elements and Ω the set of three Euler angles which relate the principal axes of the chemical shielding tensor to the arbitrary axes[1]. Unlike the dipolar Hamiltonian treatment which only had a rank 2 component, components of ranks l = 0, 1, and 2 may contribute to the shielding Hamiltonian. Since these ranks behave differently under rotations we shall write the overall shielding Hamiltonian to reflect this. Beginning with equation (4-33)

$$\boldsymbol{H}_i^{CS} = h\gamma_i B_o \sum_{l=0}^{2} \sum_{m}^{\pm l} (-1)^m A_{l,-m}^{CS2}(i) \bullet \boldsymbol{T}_{l,m}^{CS2}(i)$$

we define a chemical shielding interaction constant as

$$\xi_i^{CS} = h\gamma_i B_o \tag{4-37}$$

and expand the summation over the different ranks.

$$\boldsymbol{H}_i^{CS} = \xi_i^{CS}\left[A_{0,0}^{CS2}(i)\boldsymbol{T}_{0,0}^{CS2}(i) + \sum_{m}^{\pm 1}(-1)^m A_{1,-m}^{CS2}(i)\boldsymbol{T}_{1,m}^{CS2}(i) + \sum_{m}^{\pm 2}(-1)^m A_{2,-m}^{CS2}(i)\boldsymbol{T}_{2,m}^{CS2}(i)\right]$$

In other words we now have

$$\boldsymbol{H}_i^{CS} = \boldsymbol{H}_i^{CSI} + \boldsymbol{H}_i^{CSU} + \boldsymbol{H}_i^{CSA}. \tag{4-38}$$

There is good reason to separate these terms. The rank 0 component of the shielding Hamiltonian is rotationally invariant and called the isotropic chemical shielding Hamiltonian. In high-resolution NMR it is normally included in the static Hamiltonian $\boldsymbol{H}_0$. The rank 2 part is call the chemical shielding anisotropy Hamiltonian. In liquid systems this Hamiltonian averages to zero and thus not affect observed shielding values. It will contribute to relaxation of the system. On the other hand, in solid systems this component does not average away and will partially determine peak shapes in powder averages. The rank 1 component is the antisymmetric part of the shielding Hamiltonian. Since the antisymmetric part of the shielding tensor is difficult to measure, this part of the shielding

[1] In this instance we must be careful to express the elements $\boldsymbol{T}_{l,-m}^{CS2}$ in the same axis system as $A_{l,m}^{CS2}$. When $A^{CS2}$ is rotated in space, so must be $\boldsymbol{T}^{CS2}$. Essentially the field vector changes relative to any new coordinate system when constructing $\boldsymbol{T}^{CS2}$.

Hamiltonian is usually assumed small and neglected.

The isotropic component ($l = 0$) of the chemical shielding Hamiltonian is thus written

$$\boldsymbol{H}_i^{CSI}(AAS) = \xi_i^{CS} A_{0,0}^{CS2}(i, AAS) \bullet \boldsymbol{T}_{l,-m}^{CS2}(i, AAS) \qquad , \qquad \text{(4-39)}$$

the antisymmetric component ($l = 1$) of the chemical shielding Hamiltonian is

$$\boldsymbol{H}_i^{CSU}(AAS) = \xi_i^{CS} \sum_{m}^{\pm 1} (-1)^m A_{1,-m}^{CS2}(i, AAS) \bullet \boldsymbol{T}_{1,m}^{CS2}(i, AAS) \qquad , \qquad \text{(4-40)}$$

and the anisotropic component ($l = 2$) of the chemical shielding Hamiltonian is

$$\boldsymbol{H}_i^{CSA}(AAS) = \xi_i^{CS} \sum_{m}^{\pm 2} (-1)^m A_{2,m}^{CS2}(i, AAS) \bullet \boldsymbol{T}_{2,-m}^{CS2}(i, AAS) \qquad \text{(4-41)}$$

Throughout GAMMA, we desire all rank 2 spatial tensor irreducible spherical components to be similar to rank 2 normalized spherical harmonics if possible. Thus, we here scale the shielding spatial tensor such that the components $A_{2,m}^{CS2}$ will become normalized rank two spherical harmonics when the asymmetry term is zero, $\eta^{CS} = 0$. Thus our aim is to use the following spherical tensor to define the spatial chemical shielding tensor.

$$[A_{2,m}^{CSA} = KA_{2,m}^{CS2}]_{\eta = 0} = Y_{2,m} \qquad \text{(4-42)}$$

Now application of equation (4-42) on the $l = 2$, $m = 2$ component reveals the value of the constant K

$$A_{2,0}^{CSA}\Big|_{\eta = 0} = KA_{2,0}^{CS2} = K\sqrt{3/2}\,\delta_{zz} = Y_{2,0}\Big|_{\substack{\theta = 0 \\ \varphi = 0}} = \sqrt{\frac{5}{4\pi}}$$

$$K\sqrt{3/2}\,\delta_{zz} = \sqrt{\frac{5}{4\pi}} \qquad K = \sqrt{\frac{5}{6\pi}\delta_{zz}^{-1}}$$

and our scaled chemical shielding spatial tensor is then

$$A_{2,m}^{CSA} = \sqrt{\frac{5}{6\pi}}\,\delta_{zz}^{-1} A_{l,m}^{CS2}$$

The (rank 2) components $A_{l,m}^{CSA}$ in the principal axis system are

$$A_{2,0}^{CSA}(PAS) = \sqrt{\frac{5}{4\pi}} \qquad A_{2,\pm 1}^{CSA}(PAS) = 0 \qquad A_{2,\pm 2}^{CSA}(PAS) = \sqrt{\frac{5}{24\pi}}\eta$$

The anisotropic component ($l = 2$) of the chemical shielding Hamiltonian, equation (4-41), is then equivalently expressed by

$$\boldsymbol{H}_i^{CSA}(AAS) = \xi_i^{CSA}\sum_{m}^{\pm 2}(-1)^m A_{2,m}^{CSA}(i, AAS)\bullet \boldsymbol{T}_{2,-m}^{CS2}(i, AAS) \tag{4-43}$$

where we have define the chemical shielding anisotropy interaction constant to take into account that we have scaled the spatial tensor components by the factor $K^{-1}$.

$$\xi_i^{CSA} = \xi_i^{CSA}K_i = \sqrt{\frac{6\pi}{5}}h\gamma_i B_o \delta_{zz}(i) \tag{4-44}$$

and the chemical shielding Hamiltonian for a single spin becomes

$$\boldsymbol{H}_i^{CS} = \boldsymbol{H}_i^{CSI} + \boldsymbol{H}_i^{CSU} + \boldsymbol{H}_i^{CSA}$$

$$\boldsymbol{H}_i^{CS} = \xi_i^{CS}\left[A_{00}^{CS2}(i)\boldsymbol{T}_{00}^{CS2}(i) + \sum_{m}^{\pm 1}(-1)^m A_{1-m}^{CS2}(i)\boldsymbol{T}_{1m}^{CS2}(i)\right] + \xi_i^{CSA}\sum_{m}^{\pm 2}(-1)^m A_{2-m}^{CSA}(i)\boldsymbol{T}_{2m}^{CS2}(i)$$

When working with an entire spin system one must sum over all spins with the tensors being in the same coordinate system, for our purposes the laboratory system. The chemical shielding Hamiltonian for a spin system becomes the following.

$$\boldsymbol{H}^{CS} = \sum_{i}^{spins}\boldsymbol{H}_i^{CS} = \sum_{i}^{spins}\xi_i^{CS}\sum_{l=0}^{2}\sum_{m}^{\pm l}(-1)^m A_{l,-m}^{CS2}(i, AAS)\bullet \boldsymbol{T}_{l,m}^{CS2}(i, AAS) \tag{4-45}$$

The following figures summarize the rank 2 treatment of the shielding Hamiltonian.

## *The Rank 2 Chemical Shielding Hamiltonian Summary*

$$\boldsymbol{H}^{CS}(AAS) = \sum_{i}^{spins} \boldsymbol{H}_i^{CS}(AAS) = \sum_{i}^{spins} \xi_i^{CS} \sum_{l=0}^{2} \sum_{m}^{\pm l} (-1)^m A_{l-m}^{CS2}(i, AAS) \bullet \boldsymbol{T}_{lm}^{CS2}(i, AAS)$$

$$\boldsymbol{H}_i^{CS}(AAS) = \xi_i^{CS} \sum_{l=0}^{2} \sum_{m}^{\pm 2} (-1)^m A_{l-m}^{CS2}(i, AAS) \boldsymbol{T}_{lm}^{CS2}(i, AAS)$$

$$\xi_i^{CS} = h\gamma_i B_o$$

$$A_{l,m}^{CS2}(i, AAS) = \sum_{m'}^{\pm l} D_{mm'}^{l}(\varphi, \theta, \chi) A_{l,m'}^{CS2}(i, PAS)$$

$$A_{0,0}^{CS2}(i, PAS) = -\sqrt{3}\sigma_{iso}(i)$$

$$A_{1,0}^{CS2}(i, PAS) = -\frac{i}{\sqrt{2}}[\sigma_{xy}(i, PAS) - \sigma_{yx}(i, PAS)]$$

$$A_{1,\pm 1}^{CS2}(i, PAS) = -\frac{1}{2}[(\sigma_{zx}(i, PAS) - \sigma_{xz}(i, PAS)) \pm i(\sigma_{zy}(i, PAS) - \sigma_{yz}(i, PAS))]$$

$$A_{2,0}^{CS2}(i, PAS) = \sqrt{3/2}\,\delta_{zz}(i) \qquad A_{2,\pm 1}^{CS2}(i, PAS) = 0 \qquad A_{2,\pm 2}^{CS2}(i, PAS) = \frac{1}{2}\delta_{zz}(i)\eta(i)$$

$$T_{0,0}^{CS2}(i, AAS) = \frac{-1}{\sqrt{3}}\left[\boldsymbol{I}_{iz}B_z + \frac{1}{2}(\boldsymbol{I}_{i+}B_{-} + \boldsymbol{I}_{i-}B_{+})\right] = \frac{-1}{\sqrt{3}}\vec{\boldsymbol{I}}_i \bullet \vec{\boldsymbol{B}}_n$$

$$T_{1,0}^{CS2}(i, AAS) = \frac{-1}{2\sqrt{2}}[\boldsymbol{I}_{i+}B_{-} - \boldsymbol{I}_{i-}B_{+}] \qquad T_{1,\pm 1}^{CS2}(i, AAS) = \frac{-1}{2}[\boldsymbol{I}_{i\pm}\ B_z - \boldsymbol{I}_{iz}B_{\pm}\ ]$$

$$T_{2,0}^{CS2}(i, AAS) = \frac{1}{\sqrt{6}}[3\boldsymbol{I}_{iz}B_z - (\vec{\boldsymbol{I}}_i \bullet \vec{\boldsymbol{B}}_n)]$$

$$T_{2,\pm 1}^{CS2}(i, AAS) = \mp\frac{1}{2}[\boldsymbol{I}_{i\pm}\ B_z + \boldsymbol{I}_{iz}B_{\pm}\ ] \qquad T_{2,\pm 2}^{CS2}(i, AAS) = \frac{1}{2}[\boldsymbol{I}_{i\pm}\ B_{\pm}\ ]$$

Although these equations are generally applicable, it is convenient to express the shielding Hamiltonian with clear separation between the different ranks (the components with differing values of $l$). The isotropic component $\boldsymbol{H}^{CSI}$ in the treatment of liquid samples will normally be placed into an overall isotropic Hamiltonian, $H_0$ because it does not disappear upon rotational averaging. The asymmetric component, $\boldsymbol{H}^{CSU}$, is usually zero, the shielding tensor taken as essentially symmetric. The anisotropic component, $\boldsymbol{H}^{CSA}$, will be a contributor to relaxation processes.

### *The Chemical Shielding Hamiltonian, Rank 2 Treatment*

$$\boldsymbol{H}^{CS} = \sum_{i}^{spins} \boldsymbol{H}_i^{CS}$$

$$\boldsymbol{H}_i^{CS} = \boldsymbol{H}_i^{CSI} + \boldsymbol{H}_i^{CSU} + \boldsymbol{H}_i^{CSA}$$

Isotropic Chemical Shielding

$$\boldsymbol{H}_i^{CSI}(AAS) = \xi_i^{CS} A_{0,0}^{CS2}(i, AAS)\boldsymbol{T}_{0,0}^{CS2}(i, AAS) = h\gamma_i B_o \sigma_{iso}(i)\vec{\boldsymbol{I}}_i \bullet \vec{\boldsymbol{B}}_n$$

$$\xi_i^{CS} = h\gamma_i B_o \qquad A_{0,0}^{CS2}(i, AAS) = -\sqrt{3}\sigma_{iso}(i) \qquad \boldsymbol{T}_{0,0}^{CS2}(i, AAS) = \frac{-1}{\sqrt{3}}\vec{\boldsymbol{I}}_i \bullet \vec{\boldsymbol{B}}_n$$

$$T_{0,0}^{CS2}(AAS) = \frac{-1}{2\sqrt{3}}\begin{bmatrix} B_z & B_- \\ B_+ & -B_z \end{bmatrix} \qquad T_{0,0}^{CS2}(LAB) = \frac{-1}{2\sqrt{3}}\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$$

Antisymmetric Chemical Shielding

$$\boldsymbol{H}_i^{CSU}(AAS) = \xi_i^{CS}\sum_{m}^{\pm 1}(-1)^m A_{1-m}^{CS2}(i, AAS)\boldsymbol{T}_{1m}^{CS2}(i, AAS)$$

$$A_{1,0}^{CS2}(AAS) = -\frac{i}{\sqrt{2}}[\sigma_{xy} - \sigma_{yx}] \qquad A_{1,\pm 1}^{CS2}(AAS) = -\frac{1}{2}[(\sigma_{zx} - \sigma_{xz}) \pm i(\sigma_{zy} - \sigma_{yz})]$$

$$\xi_i^{CS} = h\gamma_i B_o \qquad T_{1,0}^{CS2}(i) = \frac{-1}{2\sqrt{2}}[\boldsymbol{I}_{i+}B_- - \boldsymbol{I}_{i-}B_+] \qquad T_{1,\pm 1}^{CS2}(i) = \frac{-1}{2}[\boldsymbol{I}_{i\pm}\ B_z - \boldsymbol{I}_{iz}B_\pm\ ]$$

$$T_{1,0}^{CS2}(AAS) = \frac{-1}{2\sqrt{2}}\begin{bmatrix} 0 & B_- \\ -B_+ & 0 \end{bmatrix} \qquad T_{1,-1}^{CS2}(AAS) = \frac{-1}{2}\begin{bmatrix} -B_-/2 & 0 \\ B_z & B_-/2 \end{bmatrix} \qquad T_{1,1}^{CS2}(AAS) = \frac{-1}{2}\begin{bmatrix} -B_+/2 & B_z \\ 0 & B_+/2 \end{bmatrix}$$

$$T_{1,0}^{CS2}(LAB) = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \qquad T_{1,-1}^{CS2}(LAB) = \frac{-1}{2}\begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix} \qquad T_{1,1}^{CS2}(LAB) = \frac{-1}{2}\begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}$$

## *The Chemical Shielding Anisotropy Hamiltonian*
## *Rank 2 Treatment*

Arbitrary Axis System

$$\boldsymbol{H}_i^{CSA}(AAS) = \xi_i^{CSA}\sum_{m}^{\pm 2}(-1)^m A_{2-m}^{CSA}(i, AAS)\boldsymbol{T}_{2m}^{CS2}(i, AAS)$$

$$T_{2,0}^{CS2}(i) = \frac{1}{\sqrt{6}}[3\boldsymbol{I}_{iz}B_z - (\vec{\boldsymbol{I}}_i \bullet kkk\vec{\boldsymbol{B}}_n)]$$

$$T_{2,\pm 1}^{CS2}(i) = \mp\frac{1}{2}[\boldsymbol{I}_{i\pm}\ B_z + \boldsymbol{I}_{iz}B_{\pm}\ ] \qquad T_{2,\pm 2}^{CS2}(i, AAS) = \frac{1}{2}[\boldsymbol{I}_{i\pm}\ B_{\pm}\ ]$$

$$\xi_i^{CSA} = \sqrt{\frac{6\pi}{5}}h\gamma_i B_o \delta_{zz}(i)$$

$$A_{2,0}^{CSA}(PAS) = \sqrt{\frac{5}{4\pi}} \qquad A_{2,\pm 1}^{CSA}(PAS) = 0 \qquad A_{2,\pm 2}^{CSA}(PAS) = \sqrt{\frac{5}{24\pi}}\eta$$

$$A_{2,m}^{CSA}(i, AAS) = \sum_{m'}^{\pm 2} D_{mm'}^{2}(\varphi, \theta, \chi)A_{2,m'}^{CSA}(i, PAS)$$

Laboratory Frame

$$\boldsymbol{H}_i^{CSA}(LAB) = \xi_i^{CSA}\sum_{m}^{\pm 1}(-1)^m A_{2-m}^{CSA}(i, LAB)\boldsymbol{T}_{2m}^{CS2}(i, LAB)$$

$$T_{2,0}^{CS2}(i, LAB) = \frac{2}{\sqrt{6}}\boldsymbol{I}_{iz} \qquad T_{2,\pm 1}^{CS2}(i, LAB) = \mp\frac{1}{2}\boldsymbol{I}_{i\pm} \qquad T_{2,\pm 2}^{CS2}(i, LAB) = 0$$

$$\xi_i^{CSA} = \sqrt{\frac{6\pi}{5}}h\gamma_i B_o \delta_{zz}(i)$$

$$A_{2,0}^{CSA}(PAS) = \sqrt{\frac{5}{4\pi}} \qquad A_{2,\pm 1}^{CSA}(PAS) = 0 \qquad A_{2,\pm 2}^{CSA}(PAS) = \sqrt{\frac{5}{24\pi}}\eta$$

$$A_{2,m}^{CSA}(i, LAB) = \sum_{m'}^{\pm 2} D_{mm'}^{2}(\varphi_{PAS \to LAB}, \theta_{PAS \to LAB}, \chi_{PAS \to LAB})A_{2,m'}^{CSA}(i, PAS)$$

$$A_{2,m}^{CSA}(i, LAB)\Big|_{\eta = 0} = Y_{2,m}(\theta, \varphi)$$

$$T_{2,0}^{CS2}(LAB) = \frac{-2}{\sqrt{6}}\begin{bmatrix}1 & 0\\ 0 & -1\end{bmatrix} \qquad T_{2,1}^{CS2}(LAB) = \frac{-1}{2}\begin{bmatrix}0 & 1\\ 0 & 0\end{bmatrix} \qquad T_{2,-1}^{CS2}(LAB) = \frac{1}{2}\begin{bmatrix}0 & 0\\ 1 & 0\end{bmatrix} \qquad T_{2,\pm 2}^{CS2}(LAB) = \begin{bmatrix}0 & 0\\ 0 & 0\end{bmatrix}$$

## 8.7.5 The Dipolar Hamiltonian

The classical interaction energy between two dipoles, $\vec{\mu}_i$ and $\vec{\mu}_j$, at a distance r from one another is[1]

$$E^D_{i,j} = \frac{\vec{\mu}_i \bullet \vec{\mu}_j}{r^3_{ij}} + 3\frac{(\vec{\mu}_i \bullet \vec{\boldsymbol{r}}_{ij})(\vec{\mu}_j \bullet \vec{\boldsymbol{r}}_{ij})}{r^5_{ij}}$$

where $\vec{\mu}$ is the magnetic moment, $i$ and $j$ spin indices, $E$ the energy, and $\vec{\boldsymbol{r}}_{ij}$ the vector connecting the two spins. The superscript $D$ is used to denote a dipolar interaction. The associated Hamiltonian is obtained from substitution of $h\gamma\vec{\boldsymbol{I}}_i$ for $\vec{\mu}_i$ (here $h = 2\pi h$).

$$\boldsymbol{H}^D_{i,j} = \frac{h^2\gamma_i\gamma_j}{r^3_{ij}}\left[\vec{\boldsymbol{I}}_i \bullet \vec{\boldsymbol{I}}_j - \frac{3}{r^2_{ij}}(\vec{\boldsymbol{I}}_i \bullet \vec{\boldsymbol{r}}_{ij})(\vec{\boldsymbol{I}}_j \bullet \vec{\boldsymbol{r}}_{ij})\right]$$

Using normalized unit vectors pointing in the direction of $\vec{\boldsymbol{r}}_{ij}$, $\vec{\boldsymbol{e}}_{ij} = \vec{\boldsymbol{r}}_{ij}/r_{ij}$ , the equation becomes

$$\boldsymbol{H}^D_{i,j} = \frac{h^2\gamma_i\gamma_j}{r^3_{ij}}[\vec{\boldsymbol{I}}_i \bullet \vec{\boldsymbol{I}}_j - 3(\vec{\boldsymbol{I}}_i \bullet \vec{\boldsymbol{e}}_{ij})(\vec{\boldsymbol{I}}_j \bullet \vec{\boldsymbol{e}}_{ij})] = \frac{h^2\gamma_i\gamma_j}{r^3_{ij}}[\vec{\boldsymbol{I}}_i \bullet \mathbf{1} \bullet \vec{\boldsymbol{I}}_j - 3(\vec{\boldsymbol{I}}_i \bullet \hat{\boldsymbol{E}}_{ij} \bullet \vec{\boldsymbol{I}}_j)]$$

where $\hat{\boldsymbol{E}}_{ij}$ is the matrix formed from the dyadic product of the two $\vec{\boldsymbol{e}}_{ij}$ unit vectors. A dipolar tensor $\hat{\boldsymbol{D}}_{ij}$ between the two spins can be defined as

$$\hat{\boldsymbol{D}}_{ij} = \mathbf{1} - 3\hat{\boldsymbol{E}}_{ij}.$$

and thus the dipolar Hamiltonian for a spin pair $i$ & $j$, $\boldsymbol{H}^D_{i,j}$, given by

$$\boldsymbol{H}^D_{i,j} = \frac{h^2\gamma_i\gamma_j}{r^3_{ij}}[\vec{\boldsymbol{I}}_i \bullet \hat{\boldsymbol{D}}_{ij} \bullet \vec{\boldsymbol{I}}_j]$$

where $\vec{\boldsymbol{I}}_i$ is the spin angular momentum operator of spin $i$ and $\hat{\boldsymbol{D}}_{ij}$ the dipolar tensor between the two spins. In expanded matrix form this equation looks like

$$\boldsymbol{H}^D_{i,j} = \frac{h^2\gamma_i\gamma_j}{r^3_{ij}}\begin{bmatrix}\boldsymbol{I}_{ix} & \boldsymbol{I}_{iy} & \boldsymbol{I}_{iz}\end{bmatrix} \bullet \begin{bmatrix} D_{xx} & D_{xy} & D_{xz} \\ D_{yx} & D_{yy} & D_{yz} \\ D_{zx} & D_{zy} & D_{zz}\end{bmatrix}_{ij} \bullet \begin{bmatrix}\boldsymbol{I}_{jx} \\ \boldsymbol{I}_{jy} \\ \boldsymbol{I}_{jz}\end{bmatrix}. \tag{4-46}$$

An equivalent equation explicitly showing the matrix multiplication is

[1] See Slichter, page 66, equation (3.2).

$$\boldsymbol{H}^{D}_{i,j} = \frac{h^2\gamma_i\gamma_j}{r^3_{ij}} \sum_{u}^{axes}\sum_{v}^{axes} \langle 1|\vec{\boldsymbol{I}}_i|u\rangle\langle u|\hat{\boldsymbol{D}}_{ij}|v\rangle\langle v|\vec{\boldsymbol{I}}_j|1\rangle \tag{4-47}$$

with $u, v \in \{x, y, z\}$ . Equation (4-24) can be rearranged to produce an equation involving two rank 2 Cartesian tensors by taking the dyadic product of the vectors $\boldsymbol{I}_i$ and $\boldsymbol{I}_j$.

$$\boldsymbol{H}^{D}_{i,j} = \frac{h^2\gamma_i\gamma_j}{r^3_{ij}} \sum_{u}^{axes}\sum_{v}^{axes} \langle u|\hat{D}_{ij}|v\rangle\langle v|\vec{\boldsymbol{I}}_j|1\rangle\langle 1|\vec{\boldsymbol{I}}_i|u\rangle = \frac{h^2\gamma_i\gamma_j}{r^3_{ij}} \sum_{u}^{axes}\sum_{v}^{axes} \langle u|\hat{D}_{ij}|v\rangle\langle v|\vec{\boldsymbol{I}}_j\vec{\boldsymbol{I}}_i|u\rangle$$

The dyadic product to produce $\vec{\boldsymbol{I}}_j\vec{\boldsymbol{I}}_i$ is explicitly done *via*

$$\begin{bmatrix} \boldsymbol{I}_{jx} \\ \boldsymbol{I}_{jy} \\ \boldsymbol{I}_{jz} \end{bmatrix} \bullet \begin{bmatrix} \boldsymbol{I}_{ix} & \boldsymbol{I}_{iy} & \boldsymbol{I}_{iz} \end{bmatrix} = \begin{bmatrix} \boldsymbol{I}_{jx}\boldsymbol{I}_{xi} & \boldsymbol{I}_{jx}\boldsymbol{I}_{yi} & \boldsymbol{I}_{jx}\boldsymbol{I}_{zi} \\ \boldsymbol{I}_{jy}\boldsymbol{I}_{xi} & \boldsymbol{I}_{jy}\boldsymbol{I}_{yi} & \boldsymbol{I}_{jy}\boldsymbol{I}_{zi} \\ \boldsymbol{I}_{jz}\boldsymbol{I}_{xi} & \boldsymbol{I}_{jz}\boldsymbol{I}_{yi} & \boldsymbol{I}_{jz}\boldsymbol{I}_{zi} \end{bmatrix}.$$

and the matrix $\hat{D}_{ij}$ in the principle axis system ($\vec{\boldsymbol{e}}_{ij} = \vec{k}$) given by

$$\hat{D}_{ij}\Big|_{PAS} = 1 - 3\hat{E}_{ij}\Big|_{PAS} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} - 3\begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \bullet \begin{bmatrix} 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -2 \end{bmatrix}. \tag{4-48}$$

Letting $\hat{\boldsymbol{T}}_{ij} = \vec{\boldsymbol{I}}_j\vec{\boldsymbol{I}}_i$, the Hamiltonian is expressed as a scalar product of two rank 2 Cartesian tensors.

$$\boldsymbol{H}^{D}_{i,j} = \frac{h^2\gamma_i\gamma_j}{r^3_{ij}} \hat{D}_{ij} \bullet \hat{\boldsymbol{T}}_{ij}$$

or equivalently (4-49)

$$\boldsymbol{H}^{D}_{i,j} = \frac{h^2\gamma_i\gamma_j}{r^3_{ij}} \sum_{u}^{axes}\sum_{v}^{axes} \langle u|\hat{D}_{ij}|v\rangle\langle v|\hat{\boldsymbol{T}}_{ij}|u\rangle$$

Equation (4-49) can be rewritten in terms of irreducible spherical components rather than the current Cartesian components[1].

$$\boldsymbol{H}_{i,j}^{D} = \frac{h^2\gamma_i\gamma_j}{r_{ij}^3}\sum_{l=0}^{2}\sum_{m}^{\pm l}(-1)^m D_{lm}^{D}(ij)\boldsymbol{T}_{l-m}^{D}(ij) \tag{4-50}$$

We obtain the 9 irreducible spherical components of the dipolar spin tensor (rank 2), $\boldsymbol{T}_{l-m}^{D}(ij)$ , directly from the Cartesian components, $\langle v|\hat{\boldsymbol{T}}_{ij}|u\rangle$ , as indicated in GAMMA Class Documentation on Spin Tensor. The nomenclature used here for a tensor component is

$$\boldsymbol{T}_{l,m} ,$$

where the subscript *l* spans the rank (in this case 2) as *l* = [0, 2], and the subscript *m* spans *+/- l, m = [-l, l*]. The nine formulas for these quantities a listed in the following figure.

### *Dipolar Irreducible Spherical Spin Tensor Components*

$$T_{0,0}^{D}(ij) = \frac{-1}{\sqrt{3}}\left[I_{iz}I_{jz} + \frac{1}{2}(I_{i+}I_{j-} + I_{i-}I_{j+})\right]$$

$$T_{1,0}^{D}(ij) = \frac{-1}{2\sqrt{2}}[I_{i+}I_{j-} + I_{i-}I_{j+}] \qquad T_{1,\pm1}^{D}(ij) = \frac{-1}{2\sqrt{2}}[I_{i\pm}I_{jz} + I_{iz}I_{j\pm}]$$

$$T_{2,0}^{D}(ij) = \frac{1}{\sqrt{6}}[3I_{iz}I_{jz} - (\boldsymbol{I}_i \bullet \boldsymbol{I}_j)]$$

$$T_{2,\pm1}^{D}(ij) = \mp\frac{1}{2}[I_{i\pm}I_{jz} + I_{iz}I_{j\pm}] \qquad T_{2,\pm2}^{D}(ij) = \frac{1}{2}[I_{i\pm}I_{j\pm}]$$

The matrix representation of these nine tensor components will depend upon the matrix representations of the individual spin operators from which they are constructed[2]. These in turn depend upon the spin quantum numbers of the two spins involved. For a treatment of two spin 1/2 particles the dipolar tensor components are expressed in their matrix form (spanning the composite Hilbert space of the two spins) in the default product basis of GAMMA as follows[3]. In this case the spin indices are implicit.

[1] The purpose of this step is to place $\boldsymbol{H}_{i,j}^{D}$ in a format which facilitates rotations on its coordinate system. For a more detailed explanation see the description in Class Spin Tensor.

[2] Note that the spin tensors are invariably constructed in the laboratory coordinate system. Here the z-axis corresponds to the direction of the spectrometer static magnetic field and the coordinate system is right-handed.

[3] The GAMMA program Dip_Spin_T.cc was used to generate these matrices. It is listed at the end of this chapter.

## Dipolar Irreducible Spherical Spin Tensor Component Matrix Representations

$$T^D_{0,0} = \frac{-1}{4\sqrt{3}}\begin{bmatrix}1&0&0&0\\0&-1&2&0\\0&2&-1&0\\0&0&0&1\end{bmatrix} \qquad T^D_{1,0} = \frac{1}{2\sqrt{2}}\begin{bmatrix}0&0&0&0\\0&0&-1&0\\0&1&0&0\\0&0&0&0\end{bmatrix} \qquad T^D_{1,-1} = \frac{1}{4}\begin{bmatrix}0&0&0&0\\1&0&0&0\\-1&0&0&0\\0&1&-1&0\end{bmatrix} \qquad T^D_{1,1} = \frac{1}{4}\begin{bmatrix}0&1&-1&0\\0&0&0&1\\0&0&0&-1\\0&0&0&0\end{bmatrix}$$

### Matrix Representations in 2-spin (I=1/2) Hilbert Space

$$T^D_{2,0} = \frac{1}{2\sqrt{6}}\begin{bmatrix}1&0&0&0\\0&-1&-1&0\\0&-1&-1&0\\0&0&0&1\end{bmatrix} \quad T^D_{2,1} = \frac{1}{4}\begin{bmatrix}0&-1&-1&0\\0&0&0&1\\0&0&0&1\\0&0&0&0\end{bmatrix} \quad T^D_{2,-1} = \frac{1}{4}\begin{bmatrix}0&0&0&0\\1&0&0&0\\1&0&0&0\\0&-1&-1&0\end{bmatrix} \quad T^D_{2,-2} = \frac{1}{2}\begin{bmatrix}0&0&0&0\\0&0&0&0\\0&0&0&0\\1&0&0&0\end{bmatrix} \quad T^D_{2,2} = \frac{1}{2}\begin{bmatrix}0&0&0&1\\0&0&0&0\\0&0&0&0\\0&0&0&0\end{bmatrix}$$

The 9 irreducible spherical components of a rank two spatial tensor, $A_{lm}$, are related to its Cartesian components by the following formulas (See GAMMA Class Documentation on Spatial Tensor).

$$A_{0,0} = \frac{-1}{\sqrt{3}}[A_{xx} + A_{yy} + A_{zz}] = \frac{-1}{\sqrt{3}}Tr\{A\}$$

$$A_{1,0} = \frac{-i}{\sqrt{2}}[A_{xy} - A_{yx}] \qquad A_{1,\pm1} = \frac{-1}{2}[A_{zx} - A_{xz} \pm i(A_{zy} - A_{yz})]$$

$$A_{2,0} = \sqrt{6}[3A_{zz} - (A_{xx} + A_{yy} + A_{zz})] = \sqrt{6}[3A_{zz} - Tr\{A\}] \tag{4-51}$$

$$A_{2,\pm1} = \mp\frac{1}{2}[A_{xz} + A_{zx} \pm i(A_{yz} + A_{zy})] \qquad A_{2,\pm2} = \frac{1}{2}[A_{xx} - A_{yy} \pm i(A_{xy} + A_{yx})]$$

Again the subscript *l* spans the rank as *l* = [0, 2], and the subscript *m* spans *+/- l, m* = [*-l, l*]. In this dipolar treatment we then have components $D^D_{l,m}(ij)$ as indicated in equation (4-50). In the principal axis system (PAS) of the dipolar interaction[1] these components are obtained by substituting the Cartesian elements of $\hat{D}_{ij}$ in equation (4-48) into (4-51).

$$D^D_{0,0}(ij, PAS) = 0 \qquad D^D_{1,0}(ij, PAS) = 0 \qquad D^D_{1,\pm1}(ij, PAS) = 0$$

$$D^D_{2,0}(ij, PAS) = -\sqrt{6} \tag{4-52}$$

$$D^D_{2,\pm1}(ij, PAS) = 0 \qquad D^D_{2,\pm2}(ij, PAS) = 0$$

All but one of the spherical components is zero because the dipolar spatial tensor is symmetric and traceless.

[1] The dipolar principal axis system for a spin pair has the z-axis pointing along the vector connecting the two spins. The orientation of the x and y axes are inconsequential due to the cylindrical symmetry of the interaction about the dipole vector (PAS z-axis).

Throughout GAMMA, we desire all spatial components to be as similar to normalized spherical harmonics as possible. Thus, we here scale the dipolar spatial tensor such that the $2, 0$ component (the only non-zero one in this instance) will have a magnitude of the $m = 0$ rank two spherical harmonic when the two spherical angles are set to zero; $Y_{2,0}(\theta, \varphi)\big|_{\theta = \varphi = 0} = \sqrt{5/(4\pi)}$ .

We define a new dipolar spatial tensor such that

$$A^D_{l,m} = -\sqrt{5/(24\pi)}D^D_{l,m} \tag{4-53}$$

and rewrite the dipolar Hamiltonian given in equation (4-50) as

$$\boldsymbol{H}^D_{i,j} = \left(\frac{h^2\gamma_i\gamma_j}{r^3_{ij}}\right)\left(-\sqrt{\frac{24\pi}{5}}\right)\sum_{l=0}^{2}\sum_{m}^{\pm l}(-1)^m A^D_{lm}(ij)\boldsymbol{T}^D_{l-m}(ij)$$

$$\boldsymbol{H}^D_{i,j} = -2\sqrt{\frac{6\pi}{5}}\frac{h^2\gamma_i\gamma_j}{r^3_{ij}}\sum_{l=0}^{2}\sum_{m}^{\pm l}(-1)^m A^D_{lm}(ij)\boldsymbol{T}^D_{l-m}(ij) \tag{4-54}$$

where

$$A^D_{0,0}(ij, PAS) = A^D_{1,0}(ij, PAS) = A^D_{1,\pm 1}(ij, PAS) = 0$$

$$A^D_{2,0}(ij, PAS) = \sqrt{5/(4\pi)} \tag{4-55}$$

$$A^D_{2,\pm 1}(ij, PAS) = A^D_{2,\pm 2}(ij, PAS) = 0$$

We can express these spatial tensor components relative to any arbitrary axis system (AAS) by rotating the tensor from the principal axes to the new axes *via* the formula

$$A^D_{l,m}(ij, AAS) = \sum_{m'}^{\pm l} D^l_{mm'}(\Omega)A^D_{l,m'}(ij, PAS) \tag{4-56}$$

where $D^l_{mm'}$ are the rank $l$ Wigner rotation matrix elements and $\Omega$ the set of three Euler angles which relate the principal axes to the arbitrary axes. As is evident from equations (4-54) - (4-36), regardless of the coordinate system, *only the rank two components will contribute to the dipolar Hamiltonian*. This is now demonstrated by combining the last two equations.

$$A^D_{0,0}(ij, AAS) = A^D_{1,0}(ij, AAS) = A^D_{1,\pm 1}(ij, AAS) = 0 \tag{4-57}$$

$$A^D_{2,m}(ij, AAS) = \sum_{m'}^{\pm 2} D^l_{mm'}(\Omega)A^D_{2,m'}(ij, PAS) = D^2_{m0}(\Omega)A^D_{2,0}(ij, PAS)$$

Using now the Wigner rotation matrix element relationship

$$D^l_{m0}(\Omega) = D^l_{m0}(\varphi, \theta, \chi) = \sqrt{\frac{4\pi}{2l+1}} Y_{lm}(\theta, \varphi) \qquad (4\text{-}58)$$

we have

$$A^D_{2,\pm 1}(ij, AAS) = \sqrt{\frac{4\pi}{5}} Y_{2m}(\theta, \varphi) A^D_{2,0}(ij, PAS) = \sqrt{\frac{4\pi}{5}} Y_{2m}(\theta, \varphi) \sqrt{\frac{5}{4\pi}} = Y_{2m}(\theta, \varphi) \qquad (4\text{-}59)$$

Because only the rank 2 dipolar spatial tensors will be non-zero, it is sufficient to utilize only the irreducible spherical rank 2 components of both the spatial and spin dipolar tensors in construction of the dipolar Hamiltonian. Equation (4-54) can be rewritten as

$$\boldsymbol{H}^D_{i,j} = \xi^D_{ij} \sum_{m}^{\pm 2} (-1)^m A^D_{2-m}(ij) \boldsymbol{T}^D_{2m}(ij) \qquad (4\text{-}60)$$

where

$$\xi^D_{ij} = -2\sqrt{\frac{6\pi}{5}} \frac{h^2 \gamma_i \gamma_j}{r^3_{ij}} \qquad A^D_{2,m}(ij, PAS) = \delta_{m0} \sqrt{\frac{5}{4\pi}} \qquad A^D_{2,m}(ij, AAS) = Y_{2m}(\theta, \varphi) \qquad (4\text{-}61)$$

When working with an entire spin system one must sum over all spin pairs with the spatial tensors being placed in the same coordinate system, usually the laboratory system. The dipolar Hamiltonian for a spin system becomes the following.

$$\boldsymbol{H}^D = \sum_{i}^{spins} \sum_{i}^{spins} \xi^D_{ij} \sum_{m=-2}^{2} (-1)^m Y^D_{2-m}(\theta_{ij}, \phi_{ij}) \boldsymbol{T}^D_{2m}(ij) \qquad (4\text{-}62)$$

Here the angles $\theta_{ij}$ and $\phi_{ij}$ are the polar angles of the dipolar vector between spins i and j when written relative to the coordinate system in which $\boldsymbol{H}^D$ is expressed. $Y_{2m}(\theta, \phi)$ are the normalized rank two spherical harmonics, the superscript $D$ indicating the dipolar interaction is unnecessary but used for consistency with other Hamiltonian definitions.

Finally, rank 2 spatial tensors are commonly specified in their principal axis system by the three components; the isotropic value $A_{iso}$, the anisotropy $\Delta A$, and the asymmetry $\eta$. These are generally given by

$$A_{iso} = \frac{1}{3} Tr\{A\} \ , \qquad \Delta A = A_{zz} - \frac{1}{2}(A_{xx} + A_{yy}) = \frac{3}{2}\delta_{zz} \qquad \eta = (\delta_{xx} - \delta_{yy}) / \delta_{zz}$$

where

$$\delta_{uv} = \frac{1}{2}(A_{uv} + A_{vu} - A_{iso})$$

A set of Euler angles $\{\alpha, \beta, \gamma\}$ is normally also given to relate the spatial tensor principle axes to another coordinate system. In the dipolar Hamiltonian derivation we have instead used $\{\varphi, \theta, \chi\}$ for the Euler angle designations because, due to the dipolar tensor symmetry, we ultimately utilize only the angles $\{\varphi, \theta\}$ which are equivalent to the common angle designations in spherical coordinates. In turn the spherical harmonics are written in these coordinates.

For the dipolar spatial tensor we have

$$A_{iso}^D = 0 \qquad \Delta^D = \sqrt{15/(8\pi)} \qquad \delta_{zz}^D = \sqrt{5/(6\pi)} \qquad \eta^D = 0$$

This is perhaps more easily seen visually by examination of the matrix breakdown into these components (superscript $D$ has been added so there is an association with the dipolar spatial tensor).

$$\hat{A}^D(PAS) = -\sqrt{\frac{5}{24\pi}}\begin{bmatrix}1&0&0\\0&1&0\\0&0&-2\end{bmatrix} = \sqrt{\frac{5}{6\pi}}\begin{bmatrix}-\frac{1}{2}&0&0\\0&-\frac{1}{2}&0\\0&0&1\end{bmatrix} = \cancelto{0}{D_{iso}\begin{bmatrix}1&0&0\\0&1&0\\0&0&1\end{bmatrix}} + \delta_{zz}^D\begin{bmatrix}-\frac{1}{2}(1-\cancelto{0}{\eta^D})&0&0\\0&-\frac{1}{2}(1+\cancelto{0}{\eta^D})&0\\0&0&1\end{bmatrix}$$

The following figure contains a grouping of the applicable dipolar Hamiltonian equations.

## The Dipolar Hamiltonian Summary

General System

$$\boldsymbol{H}^D = \sum_i^{spins}\sum_j^{spins} \xi_{ij}^D \sum_{m=-2}^{2} (-1)^m Y^D_{2-m}(\theta_{ij}, \phi_{ij})\boldsymbol{T}^D_{2m}(ij)$$

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

$$Y_{2,\pm 1}(\theta, \phi) = \mp\sqrt{\frac{15}{8\pi}}\cos\theta\sin\theta e^{\pm i\phi} \qquad Y_{2,\pm 2}(\theta, \phi) = \sqrt{\frac{15}{32\pi}}\sin^2\theta e^{\pm 2i\phi}$$

$$T^D_{2,0}(ij) = \frac{1}{\sqrt{6}}[3I_{iz}I_{jz} - (\boldsymbol{I}_i \bullet \boldsymbol{I}_j)]$$

$$T^D_{2,\pm 1}(ij) = \mp\frac{1}{2}[I_{i\pm}\ I_{jz} + I_{iz}I_{j\pm}\ ] \qquad T^D_{2,\pm 2}(ij) = \frac{1}{2}[I_{i\pm}\ I_{j\pm}\ ]$$

$$\xi^D_{ij} = -2\sqrt{\frac{6\pi}{5}}\frac{h^2\gamma_i\gamma_j}{r^3_{ij}}$$

Single Dipole

$$\boldsymbol{H}^D_{i,j} = \xi^D_{ij}\sum_m^{\pm 2}(-1)^m A^D_{2-m}(ij)\boldsymbol{T}^D_{2m}(ij)$$

$$A^D_{2,m}(ij, PAS) = \delta_{m0}\sqrt{\frac{5}{4\pi}} \qquad A^D_{2,m}(ij, AAS) = Y_{2m}(\theta, \varphi)$$

$$A^D_{l,m}(ij, AAS) = \sum_{m'}^{\pm l} D^l_{mm'}(\varphi, \theta, \chi)A^D_{l,m'}(ij, PAS)$$

Spatial Tensor

$$A^D_{iso} = 0 \qquad \Delta^D = \sqrt{15/(8\pi)} \qquad \eta^D = 0$$

## 8.7.6 The Scalar Coupling Hamiltonian

There is an experimentally observed interaction between nuclear spins which are connected by one or more chemical bonds. The strength of the interaction attenuates as the number of bonds separating the spins increases, this being due to the spin state information of one spin somehow being transmitted through the bonding electrons of the molecule and affecting the spin state of the other. Because of its seemingly isotropic nature[1] (molecular orientation does not appear of consequence) this is called scalar coupling (or J coupling, spin-spin coupling, or indirect dipolar coupling). The energy involved in the scalar coupling of two dipoles, $\vec{\mu}_i$ and $\vec{\mu}_j$, at a distance r from one another is can be represented by

$$E^J_{i,j} = \frac{\vec{\mu}_i \bullet \vec{\mu}'_j}{r^3_{ij}}$$

where $\vec{\mu}$ is the magnetic moment, $i$ and $j$ spin indices, $E$ the energy, and $\vec{\boldsymbol{r}}_{ij}$ the vector connecting the two spins. The superscript $J$ is used to denote a scalar coupling interaction. Note that the second moment has been labeled with a prime to indicate that there is some effective magnetic moment for the spin $j$ which affects spin $i$ through scalar coupling, and we can surmise that there is some matrix, say $J'$ which scales the interaction properly.

$$\hat{J}'(ij) \bullet \vec{\mu}_j = \frac{\vec{\mu}'_j}{r^3_{ij}}$$

Thus we can write the energy as

$$E^J_{i,j} = \vec{\mu}_i \bullet \hat{J}'(ij) \bullet \vec{\mu}_j$$

and replacing $\vec{\mu}_i$ with $h\gamma\vec{\boldsymbol{I}}_i$ (here $h = 2\pi h$) we obtain one form for the associated scalar coupling Hamiltonian.

$$\boldsymbol{H}^J_{i,j} = h^2\gamma_i\gamma_j[\vec{\boldsymbol{I}}_i \bullet \hat{J}'(ij) \bullet \vec{\boldsymbol{I}}_j]$$

Traditionally scalar coupling is measured in Hertz, so the proper equation in NMR is

$$\boldsymbol{H}^J_{i,j} = h[\vec{\boldsymbol{I}}_i \bullet \hat{J}(ij) \bullet \vec{\boldsymbol{I}}_j]$$

We have treated the interaction as if it is orientationally dependent by constructing a scalar cou-

[1] In this section we apply the usual tensor formalism to define the scalar coupling Hamiltonian, although the isotropic part will normally be sufficient for its description. The full tensor treatment is performed, partly for consistency with other Hamiltonian definitions and party because of the scalar relaxation which may occur if $\boldsymbol{H}^J_{i,j}$ were time dependent. We will later argue that scalar relaxation can be accounted for by either chemical exchange processes or by more appropriate relaxation mechanisms. In this case, overall molecular motion and associated rotational correlation times do not play a central role in scalar relaxation; this is reasonable as the bonding electrons responsible for the interactions should not be affected by molecular orientation.

pling tensor (or matrix). Furthermore we have included all constants and the interaction scaling factors, except for Plank’s constant, into $\hat{J}$ whose elements are in Hertz. In expanded matrix form this equation looks like

$$\boldsymbol{H}^{J}_{i,j} = h\begin{bmatrix}\boldsymbol{I}_{ix} & \boldsymbol{I}_{iy} & \boldsymbol{I}_{iz}\end{bmatrix} \bullet \begin{bmatrix} J_{xx} & J_{xy} & J_{xz} \\ J_{yx} & J_{yy} & J_{yz} \\ J_{zx} & J_{zy} & J_{zz}\end{bmatrix}_{ij} \bullet \begin{bmatrix}\boldsymbol{I}_{jx} \\ \boldsymbol{I}_{jy} \\ \boldsymbol{I}_{jz}\end{bmatrix}. \qquad (4\text{-}63)$$

An equivalent equation explicitly showing the matrix multiplication is

$$\boldsymbol{H}^{J}_{i,j} = h\sum_{u}^{axes}\sum_{v}^{axes}\langle 1|\vec{\boldsymbol{I}}_{i}|u\rangle\langle u|\hat{\boldsymbol{J}}_{ij}|v\rangle\langle v|\vec{\boldsymbol{I}}_{j}|1\rangle \qquad (4\text{-}64)$$

with $u, v \in \{x, y, z\}$ . Equation (4-64) can be rearranged to produce an equation involving two rank 2 Cartesian tensors by taking the dyadic product of the vectors $\boldsymbol{I}_i$ and $\boldsymbol{I}_j$.

$$\boldsymbol{H}^{J}_{i,j} = h\sum_{u}^{axes}\sum_{v}^{axes}\langle u|\hat{\boldsymbol{J}}_{ij}|v\rangle\langle v|\vec{\boldsymbol{I}}_{j}|1\rangle\langle 1|\vec{\boldsymbol{I}}_{i}|u\rangle = h\sum_{u}^{axes}\sum_{v}^{axes}\langle u|\hat{\boldsymbol{J}}_{ij}|v\rangle\langle v|\vec{\boldsymbol{I}}_{j}\vec{\boldsymbol{I}}_{i}|u\rangle$$

The dyadic product to produce $\vec{\boldsymbol{I}}_j\vec{\boldsymbol{I}}_i$ is explicitly done *via*

$$\begin{bmatrix}\boldsymbol{I}_{jx} \\ \boldsymbol{I}_{jy} \\ \boldsymbol{I}_{jz}\end{bmatrix} \bullet \begin{bmatrix}\boldsymbol{I}_{ix} & \boldsymbol{I}_{iy} & \boldsymbol{I}_{iz}\end{bmatrix} = \begin{bmatrix}\boldsymbol{I}_{jx}\boldsymbol{I}_{xi} & \boldsymbol{I}_{jx}\boldsymbol{I}_{yi} & \boldsymbol{I}_{jx}\boldsymbol{I}_{zi} \\ \boldsymbol{I}_{jy}\boldsymbol{I}_{xi} & \boldsymbol{I}_{jy}\boldsymbol{I}_{yi} & \boldsymbol{I}_{jy}\boldsymbol{I}_{zi} \\ \boldsymbol{I}_{jz}\boldsymbol{I}_{xi} & \boldsymbol{I}_{jz}\boldsymbol{I}_{yi} & \boldsymbol{I}_{jz}\boldsymbol{I}_{zi}\end{bmatrix}.$$

Letting $\hat{\boldsymbol{T}}_{ij} = \vec{\boldsymbol{I}}_j\vec{\boldsymbol{I}}_i$, the Hamiltonian is expressed as a scalar product of two rank 2 Cartesian tensors.

$$\boldsymbol{H}^{J}_{i,j} = h\hat{\boldsymbol{J}}_{ij} \bullet \hat{\boldsymbol{T}}_{ij}$$

or equivalently (4-65)

$$\boldsymbol{H}^{J}_{i,j} = h\sum_{u}^{axes}\sum_{v}^{axes}\langle u|\hat{\boldsymbol{J}}_{ij}|v\rangle\langle v|\hat{\boldsymbol{T}}_{ij}|u\rangle$$

Equation (4-65) can be rewritten in terms of irreducible spherical components rather than the current Cartesian components[1].

$$\boldsymbol{H}^J_{i,j} = h\sum_{l=0}^{2}\sum_{m}^{\pm l}(-1)^m \boldsymbol{J}^J_{l-m}(ij)\boldsymbol{T}^J_{lm}(ij) \qquad (4\text{-}66)$$

We obtain the 9 irreducible spherical components of the scalar coupling spin tensor (rank 2), $\boldsymbol{T}^J_{lm}(ij)$ , directly from the Cartesian components, $\langle v|\boldsymbol{T}_{ij}|u\rangle$ , as indicated in GAMMA Class Documentation on Spin Tensor. The nomenclature used here for a tensor component is

$$\boldsymbol{T}_{l,m},$$

where the subscript *l* spans the rank (in this case 2) as *l* = [0, 2], and the subscript *m* spans *+/- l, m = [-l, l]*. The nine formulas for these quantities a listed in the following figure.

***Scalar Coupling Irreducible Spherical Spin Tensor Components***

$$T^J_{0,0}(ij) = \frac{-1}{\sqrt{3}}\left[I_{iz}I_{jz} + \frac{1}{2}(I_{i+}I_{j-} + I_{i-}I_{j+})\right]$$

$$T^J_{1,0}(ij) = \frac{-1}{2\sqrt{2}}[I_{i+}I_{j-} + I_{i-}I_{j+}] \qquad T^J_{1,\pm1}(ij) = \frac{-1}{2\sqrt{2}}[I_{i\pm}I_{jz} + I_{iz}I_{j\pm}]$$

$$T^J_{2,0}(ij) = \frac{1}{\sqrt{6}}[3I_{iz}I_{jz} - (\boldsymbol{I}_i \bullet \boldsymbol{I}_j)]$$

$$T^J_{2,\pm1}(ij) = \mp\frac{1}{2}[I_{i\pm}I_{jz} + I_{iz}I_{j\pm}] \qquad T^J_{2,\pm2}(ij) = \frac{1}{2}[I_{i\pm}I_{j\pm}]$$

The matrix representation of these nine tensor components will depend upon the matrix representations of the individual spin operators from which they are constructed[2]. These in turn depend upon the spin quantum numbers of the two spins involved. For a treatment of two spin 1/2 particles the scalar coupling spin tensor components are expressed in their matrix form (spanning the composite Hilbert space of the two spins) in the default product basis of GAMMA as follows[3]. In this case the spin indices are implicit.

[1] The purpose of this step is to place $\boldsymbol{H}^D_{i,j}$ in a format which facilitates rotations on its coordinate system. For a more detailed explanation see the description in Class Spin Tensor.

[2] Note that the spin tensors are invariably constructed in the laboratory coordinate system. Here the z-axis corresponds to the direction of the spectrometer static magnetic field and the coordinate system is right-handed.

[3] The GAMMA program J_Spin_T.cc was used to generate these matrices. It is listed at the end of this chapter.

## *Scalar Coupling Irreducible Spherical Spin Tensor Components*

### *Matrix Representations in 2-spin (I=1/2) Hilbert Space*

$$T^J_{0,0} = \frac{-1}{4\sqrt{3}}\begin{bmatrix}1&0&0&0\\0&-1&2&0\\0&2&-1&0\\0&0&0&1\end{bmatrix} \qquad T^J_{1,0} = \frac{1}{2\sqrt{2}}\begin{bmatrix}0&0&0&0\\0&0&-1&0\\0&1&0&0\\0&0&0&0\end{bmatrix} \qquad T^J_{1,-1} = \frac{1}{4}\begin{bmatrix}0&0&0&0\\1&0&0&0\\-1&0&0&0\\0&1&-1&0\end{bmatrix} \qquad T^J_{1,1} = \frac{1}{4}\begin{bmatrix}0&1&-1&0\\0&0&0&1\\0&0&0&-1\\0&0&0&0\end{bmatrix}$$

$$T^J_{2,0} = \frac{1}{2\sqrt{6}}\begin{bmatrix}1&0&0&0\\0&-1&-1&0\\0&-1&-1&0\\0&0&0&1\end{bmatrix} \quad T^J_{2,1} = \frac{1}{4}\begin{bmatrix}0&-1&-1&0\\0&0&0&1\\0&0&0&1\\0&0&0&0\end{bmatrix} \quad T^J_{2,-1} = \frac{1}{4}\begin{bmatrix}0&0&0&0\\1&0&0&0\\1&0&0&0\\0&-1&-1&0\end{bmatrix} \quad T^J_{2,-2} = \frac{1}{2}\begin{bmatrix}0&0&0&0\\0&0&0&0\\0&0&0&0\\1&0&0&0\end{bmatrix} \quad T^J_{2,2} = \frac{1}{2}\begin{bmatrix}0&0&0&1\\0&0&0&0\\0&0&0&0\\0&0&0&0\end{bmatrix}$$

Again the subscript *l* spans the rank as l = [0, 2], and the subscript m spans +/- l, m = [-l, l]. The 9 irreducible spherical components of a rank two spatial tensor, $A_{lm}$, are related to its Cartesian components by the following formulas (See GAMMA Class Documentation on Spatial Tensor).

$$A_{0,0} = \frac{-1}{\sqrt{3}}[A_{xx} + A_{yy} + A_{zz}] = \frac{-1}{\sqrt{3}}Tr\{A\}$$

$$A_{1,0} = \frac{-i}{\sqrt{2}}[A_{xy} - A_{yx}] \qquad A_{1,\pm1} = \frac{-1}{2}[A_{zx} - A_{xz} \pm i(A_{zy} - A_{yz})]$$

$$A_{2,0} = \sqrt{6}[3A_{zz} - (A_{xx} + A_{yy} + A_{zz})] = \sqrt{6}[3A_{zz} - Tr\{A\}] \tag{4-67}$$

$$A_{2,\pm1} = \mp\frac{1}{2}[A_{xz} + A_{zx} \pm i(A_{yz} + A_{zy})] \qquad A_{2,\pm2} = \frac{1}{2}[A_{xx} - A_{yy} \pm i(A_{xy} + A_{yx})]$$

Again the subscript *l* spans the rank as *l* = [0, 2], and the subscript *m* spans +/- *l*, *m* = [-*l*, *l*]. We continue to treat the spatial tensor $\hat{\boldsymbol{J}}_{ij}$ as if it may be orientationally dependent and its irreducible spherical tensor components can be obtained by substituting the Cartesian elements of $\hat{\boldsymbol{J}}_{ij}$ into equations (4-67). A general rank two Cartesian tensor can be rewritten in terms of a sum over tensors of ranks 0 through 2 as follows,

$$\hat{\boldsymbol{J}}_{ij} = \begin{bmatrix}J_{xx}&J_{xy}&J_{xz}\\J_{yx}&J_{yy}&J_{yz}\\J_{zx}&J_{zy}&J_{zz}\end{bmatrix}_i = \underbrace{J_{iso}(ij)\begin{bmatrix}1&0&0\\0&1&0\\0&0&1\end{bmatrix}}^{\text{Rank 0}} + \overbrace{\begin{bmatrix}0&\alpha_{xy}&\alpha_{xz}\\-\alpha_{xy}&0&\alpha_{yz}\\-\alpha_{xz}&-\alpha_{yz}&0\end{bmatrix}_i}^{\text{Rank 1}} + \overbrace{\begin{bmatrix}\delta_{xx}&\delta_{xy}&\delta_{xz}\\\delta_{yx}&\delta_{yy}&\delta_{yz}\\\delta_{zx}&\delta_{zy}&\delta_{zz}\end{bmatrix}_i}^{\text{Rank 2}}$$

where

$$J_{iso} = \frac{1}{3}Tr\{\hat{\boldsymbol{J}}\} \qquad \alpha_{xy} = \frac{1}{2}(J_{xy} - J_{yx}) \qquad \delta_{xy} = \frac{1}{2}(J_{xy} + J_{yx} - 2J_{iso})$$

The rank 0 part is isotropic (scalar), the rank 1 part is antisymmetric and traceless, and the rank 2 part traceless and symmetric. As with all spatial tensors, the scalar coupling spatial tensor can be specified most readily in its principal axis system, the respective principal axes would normally be different for each spin pair. The scalar coupling tensor values could in principle be experimentally determined in the tensor principal axes, and then less values are necessary to specify the tensor because many more tensor elements are zero.

$$\hat{\boldsymbol{J}}_{ij}(PAS) = J_{iso}(ij)\begin{bmatrix}1&0&0\\0&1&0\\0&0&1\end{bmatrix} + \begin{bmatrix}0&\alpha_{xy}&\alpha_{xz}\\-\alpha_{xy}&0&\alpha_{yz}\\-\alpha_{xz}&-\alpha_{yz}&0\end{bmatrix}_i + \begin{bmatrix}\delta_{xx}&0&0\\0&\delta_{yy}&0\\0&0&\delta_{zz}\end{bmatrix}_i$$

Rank 2 spatial tensors are commonly specified in their principal axis system by the three components; the isotropic value $A_{iso}$, the anisotropy $\Delta A$, and the asymmetry $\eta$. These are generally given by

$$A_{iso} = \frac{1}{3}Tr\{A\}\ , \qquad \Delta A = A_{zz} - \frac{1}{2}(A_{xx} + A_{yy}) = \frac{3}{2}\delta_{zz} \qquad \eta = (\delta_{xx} - \delta_{yy})/\delta_{zz}$$

A set of Euler angles $\{\alpha, \beta, \gamma\}$ is normally also given to relate the spatial tensor principle axes to another coordinate system. For the scalar coupling spatial tensor we have

$$\hat{\boldsymbol{J}}_{ij}(PAS) = J_{iso}(ij)\begin{bmatrix}1&0&0\\0&1&0\\0&0&1\end{bmatrix} + \begin{bmatrix}0&\alpha_{xy}&\alpha_{xz}\\-\alpha_{xy}&0&\alpha_{yz}\\-\alpha_{xz}&-\alpha_{yz}&0\end{bmatrix}_i + \delta_{zz}\begin{bmatrix}-\frac{1}{2}(1-\eta)&0&0\\0&-\frac{1}{2}(1+\eta)&0\\0&0&1\end{bmatrix}_i \tag{4-68}$$

We will now write the irreducible spherical elements of the scalar coupling tensor as, $A^J_{2,m}$. These are obtained, in the principal axis system, by placement of (4-35) into (4-67),

***Scalar Coupling Irreducible Spherical Spatial Tensor Components***
***Principal Axis System (PAS)***

$$A^J_{0,0}(PAS) = -\sqrt{3}J_{iso}$$

$$A^J_{1,0}(PAS) = -\frac{i}{\sqrt{2}}[J_{xy} - J_{yx}] \qquad A^J_{1,\pm1}2(PAS) = -\frac{1}{2}[(J_{zx} - J_{xz}) \pm i(J_{zy} - J_{yz})]$$

$$A^J_{2,0}(PAS) = \sqrt{3/2}\delta_{zz} \qquad A^J_{2,\pm1}(PAS) = 0 \qquad A^J_{2,\pm2}(PAS) = \frac{1}{2}\delta_{zz}\eta$$

We can express the spatial tensor components $A^J_{l,m}$ relative to any arbitrary axis system (AAS) by a rotation from the principal axes to the new axes *via* the formula

$$A^J_{l,m}(ij, AAS) = \sum_{m'}^{\pm l} D^l_{mm'}(\Omega) A^J_{l,m'}(ij, PAS) \qquad (4\text{-}69)$$

where $D^l_{mm'}$ are the rank $l$ Wigner rotation matrix elements and $\Omega$ the set of three Euler angles which relate the principal axes of the scalar coupling tensor to the arbitrary axes. The components of ranks l = 0, 1, and 2 could all, in theory, contribute to the scalar coupling Hamiltonian. Since these ranks behave differently under rotations we shall write the overall coupling Hamiltonian to reflect this. Beginning with equation (4-66)

$$\boldsymbol{H}^J_{ij} = h \sum_{l=0}^{2} \sum_{m}^{\pm l} (-1)^m A^J_{l,-m}(ij) \bullet \boldsymbol{T}^J_{l,m}(ij)$$

we define a chemical shielding interaction constant as

$$\xi^J_{ij} = h \qquad (4\text{-}70)$$

and expand the summation over the different ranks.

$$\boldsymbol{H}^J_{ij} = \xi_{ij}\left[A^J_{0,0}(ij)\boldsymbol{T}^J_{0,0}(ij) + \sum_{m}^{\pm 1}(-1)^m A^J_{1,-m}(ij)\boldsymbol{T}^J_{1,m}(ij) + \sum_{m}^{\pm 2}(-1)^m A^J_{2,-m}(ij)\boldsymbol{T}^J_{2,m}(ij)\right]$$

In other words we now have

$$\boldsymbol{H}^J_{ij} = \boldsymbol{H}^{JI}_i + \boldsymbol{H}^{JU}_i + \boldsymbol{H}^{JA}_i. \qquad (4\text{-}71)$$

There is good reason to separate these terms. The rank 0 component of the Hamiltonian is rotationally invariant and called the isotropic scalar coupling Hamiltonian. In high-resolution NMR it is normally included in the static Hamiltonian $\boldsymbol{H}_0$. The rank 2 part is the anisotropic part of the Hamiltonian and in liquid systems averages to zero and thus not does affect observed coupling values. It would, if it were large enough, contribute to relaxation of the system but this has not been observed. In solid systems this component does not average away and would contribute to line broadening if large enough. Again, this is too small to be observed. The rank 1 component is the antisymmetric part of the Hamiltonian. Since the antisymmetric part of the scalar coupling Hamiltonian is normally smaller than the anisotropic part, this contribution to the Hamiltonian is neglected as well. Thus by and large, we have simply

$$\boldsymbol{H}_{ij}^{J} \approx \boldsymbol{H}_{i}^{JI} + \cancelto{0}{\boldsymbol{H}_{i}^{JU}} + \cancelto{0}{\boldsymbol{H}_{i}^{JA}} = \boldsymbol{H}_{i}^{JI}. \tag{4-72}$$

How then does scalar coupling provide a relaxation mechanism? There are two types of “scalar relaxation” discussed in the literature; a.) Scalar Coupling of the 1st Kind - the coupling tensor fluctuates in time due to chemical exchange or some other changes that affect chemical bonding. and b.) Scalar Coupling of the 2nd Kind - the spin state of one coupled nucleus is fluctuating at frequencies which can “relax” the second spin of a spin pair.

For scalar relaxation of the 1st kind, a typical example would be the disappearance of the proton multiplet due to exchange in an amine group. Obviously as the protons are exchanging, either with themselves or with solvent protons, the scalar coupling tensor is being modulated in time. The scalar coupling Hamiltonian (shown below as initially written) is then time dependent

$$\boldsymbol{H}_{i,j}^{J}(t) = h[\vec{\boldsymbol{I}}_i \bullet \hat{J}(ij,t) \bullet \vec{\boldsymbol{I}}_j]$$

but the time dependency is based not on molecular reorientation (as are several other NMR related Hamiltonians) because only the isotropic part of the tensor exists. Its time dependency is characterized by an exchange rate rather than a correlation time. Thus, “scalar” relaxation of the 1st kind is one and the same as exchange effects.

An example of scalar relaxation of the second kind would be the multiplet collapse of a nucleus scalar coupled to a quadrupolar nucleus ($^{14}$N, $^{35}$Cl, $^{79}$Br, $^{2}$H, etc.). The argument is that as the spin state of the quadrupole is changing in time (due to quadrupolar relaxation) the second spin’s state is modulated and this is a “scalar relaxation effect”. This is somewhat of a misnomer as the scalar coupling Hamiltonian does not become time dependent, regardless of the spin states of either nuclei involved! The scalar coupling Hamiltonian (as initially written) in this case remains static.

$$\boldsymbol{H}_{i,j}^{J} = h[\vec{\boldsymbol{I}}_i \bullet \hat{J}(ij) \bullet \vec{\boldsymbol{I}}_j]$$

It is the system itself - described either by particular spin(s), wavefunction(s), or the density matrix - which are time dependent. Thus, it is some other Hamiltonian(s) which are causing the spin state(s) to change which are actually responsible for “scalar” relaxation of the 2nd kind[1].

The isotropic component ($l = 0$) of the scalar coupling Hamiltonian is thus written

$$\boldsymbol{H}_{ij}^{JI}(AAS) = \xi_{ij}^{J} A_{0,0}^{J}(ij, AAS) \bullet \boldsymbol{T}_{l,-m}^{J}(ij, AAS) \ , \tag{4-73}$$

the antisymmetric component ($l = 1$) of the scalar coupling Hamiltonian is

[1] This does not prevent one from writing down formulas for the relaxation of spin i which is coupled to spin j where the spin state of j is being constantly changed by some mechanism (quadrupolar relaxation, an applied rf-field at the spins resonance frequency, etc.) One simply needs to be careful in defining exactly what is producing the “scalar relaxation”. In a Redfield approach, scalar relaxation of the second type is accounted for completely by other relaxation effects. These “scalar relaxation” effects will naturally arise from the spin coupling and relaxation already specified for the system.

$$\boldsymbol{H}_{ij}^{JU}(AAS) = \xi_{ij}^{J}\sum_{m}^{\pm 1}(-1)^m A_{1,-m}^{J}(ij, AAS)\bullet \boldsymbol{T}_{1,m}^{J}(ij, AAS) \qquad , \tag{4-74}$$

and the anisotropic component ($l = 2$) of the scalar coupling Hamiltonian is

$$\boldsymbol{H}_{ij}^{J}(AAS) = \xi_{ij}^{J}\sum_{m}^{\pm 2}(-1)^m A_{2,m}^{J}(ij, AAS)\bullet \boldsymbol{T}_{2,-m}^{J}(ij, AAS) \tag{4-75}$$

Throughout GAMMA, we desire all rank 2 spatial tensor irreducible spherical components to be similar to rank 2 normalized spherical harmonics if possible. Thus, we here scale the shielding spatial tensor such that the components $A_{2,m}^{CS2}$ will become normalized rank two spherical harmonics when the asymmetry term is zero, $\eta^{CS} = 0$. Thus our aim is to use the following spherical tensor to define the spatial chemical shielding tensor.

$$[A_{2,m}^{CSA} = KA_{2,m}^{CS2}]_{\eta = 0} = Y_{2,m} \tag{4-76}$$

Now application of equation (4-42) on the $l = 2$, $m = 2$ component reveals the value of the constant K

$$A_{2,0}^{CSA}\Big|_{\eta = 0} = KA_{2,0}^{CS2} = K\sqrt{3/2}\,\delta_{zz} = Y_{2,0}\Big|_{\substack{\theta = 0\\ \varphi = 0}} = \sqrt{\frac{5}{4\pi}}$$

$$K\sqrt{3/2}\,\delta_{zz} = \sqrt{\frac{5}{4\pi}} \qquad K = \sqrt{\frac{5}{6\pi}\delta_{zz}^{-1}}$$

and our scaled chemical shielding spatial tensor is then

$$A_{2,m}^{CSA} = \sqrt{\frac{5}{6\pi}}\delta_{zz}^{-1}A_{l,m}^{CS2}$$

The (rank 2) components $A_{l,m}^{CSA}$ in the principal axis system are

$$A_{2,0}^{CSA}(PAS) = \sqrt{\frac{5}{4\pi}} \qquad A_{2,\pm 1}^{CSA}(PAS) = 0 \qquad A_{2,\pm 2}^{CSA}(PAS) = \sqrt{\frac{5}{24\pi}}\eta$$

The anisotropic component ($l = 2$) of the chemical shielding Hamiltonian, equation (4-41), is then equivalently expressed by

$$\boldsymbol{H}_{i}^{CSA}(AAS) = \xi_{i}^{CSA}\sum_{m}^{\pm 2}(-1)^m A_{2,m}^{CSA}(i, AAS)\bullet \boldsymbol{T}_{2,-m}^{CS2}(i, AAS) \tag{4-77}$$

where we have define the chemical shielding anisotropy interaction constant to take into account that we have scaled the spatial tensor components by the factor $K^{-1}$.

$$\xi_i^{CSA} = \xi_i^{CSA} K_i = \sqrt{\frac{6\pi}{5}} h\gamma_i B_o \delta_{zz}(i) \qquad (4\text{-}78)$$

and the scalar coupling Hamiltonian for a single spin pair becomes

$$\boldsymbol{H}_{ij}^{J} = \boldsymbol{H}_{ij}^{JI} + \boldsymbol{H}_{ij}^{JU} + \boldsymbol{H}_{ij}^{JA}$$

$$\boldsymbol{H}_{ij}^{J} = \xi_{ij}^{J}\left[A_{00}^{J}(ij)\boldsymbol{T}_{00}^{J}(ij) + \sum_{m}^{\pm 1}(-1)^m A_{1-m}^{J}(ij)\boldsymbol{T}_{1m}^{J}(ij)\right] + \xi_{ij}^{JA}\sum_{m}^{\pm 2}(-1)^m A_{2-m}^{JA}(ij)\boldsymbol{T}_{2m}^{J}(ij)$$

When working with an entire spin system one must sum over all spin-spin pairs with the tensors being in the same coordinate system, for our purposes the laboratory system. The scalar coupling Hamiltonian for a spin system becomes the following.

$$\boldsymbol{H}^{J} = \sum_{i}^{spins}\sum_{j>i}^{spins}\boldsymbol{H}_{ij}^{J} = \sum_{i}^{spins}\sum_{j>i}^{spins}\xi_{ij}^{J}\sum_{l=0}^{2}\sum_{m}^{\pm l}(-1)^m A_{l,-m}^{J}(ij, AAS) \bullet \boldsymbol{T}_{l,m}^{J}(ij, AAS) \qquad (4\text{-}79)$$

The following figures summarize the rank 2 treatment of the scalar coupling Hamiltonian.

## The Scalar Coupling Hamiltonian

General System

$$\boldsymbol{H}^J = \sum_{i}^{spins}\sum_{j>i}^{spins}\boldsymbol{H}_{ij}^J = \sum_{i}^{spins}\sum_{j>i}^{spins}\xi_{ij}^J\sum_{l=0}^{2}\sum_{m}^{\pm l}(-1)^m A_{l,-m}^J(ij,AAS)\bullet\boldsymbol{T}_{l,m}^J(ij,AAS)$$

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

$$Y_{2,\pm1}(\theta,\phi) = \mp\sqrt{\frac{15}{8\pi}}\cos\theta\sin\theta e^{\pm i\phi} \qquad Y_{2,\pm2}(\theta,\phi) = \sqrt{\frac{15}{32\pi}}\sin^2\theta e^{\pm 2i\phi}$$

$$T_{2,0}^D(ij) = \frac{1}{\sqrt{6}}[3I_{iz}I_{jz} - (\boldsymbol{I}_i\bullet\boldsymbol{I}_j)]$$

$$T_{2,\pm1}^D(ij) = \mp\frac{1}{2}[I_{i\pm}\ I_{jz} + I_{iz}I_{j\pm}\ ] \qquad T_{2,\pm2}^D(ij) = \frac{1}{2}[I_{i\pm}\ I_{j\pm}\ ]$$

$$\xi_{ij}^D = -2\sqrt{\frac{6\pi}{5}}\frac{h^2\gamma_i\gamma_j}{r_{ij}^3}$$

Single Spin Pair

$$\boldsymbol{H}_{ij}^J = \xi_{ij}^J\sum_{l=0}^{2}\sum_{m}^{\pm l}(-1)^m A_{l,-m}^J(ij,AAS)\bullet\boldsymbol{T}_{l,m}^J(ij,AAS)$$

$$A_{2,m}^D(ij,PAS) = \delta_{m0}\sqrt{\frac{5}{4\pi}} \qquad A_{2,m}^D(ij,AAS) = Y_{2m}(\theta,\varphi)$$

$$A_{l,m}^D(ij,AAS) = \sum_{m'}^{\pm l}D_{mm'}^l(\varphi,\theta,\chi)A_{l,m'}^D(ij,PAS)$$

Spatial Tensor

$$A_{iso}^D = 0 \qquad \Delta^D = \sqrt{15/(8\pi)} \qquad \eta^D = 0$$

## 8.7.7 The Quadrupole Hamiltonian

Nuclei having spin quantum numbers $I > 1/2$ may exhibit electric quadrupole moments, $e\boldsymbol{Q}$. Quadrupole moments will interact with any electric field gradients, $eq$, such as those due to the electron cloud surrounding the nucleus. There are many different ways to treat the associated interaction, here we follow the presentation in Slichter[1]. We start with the classical electrostatic interaction between a charge distribution $\rho(\vec{r})$ (in this case the nucleus) and an electrostatic potential $V(\vec{r})$ (set up by the surrounding electron cloud). The associated electrostatic energy is given by

$$E_e = \int_{all\ space} \rho(\vec{r})V(\vec{r})d^3r$$

In this case $V(\vec{r})$ is the potential produced by the electron cloud at the point $\vec{r}$ in the nucleus. Expansion of the electrostatic potential as a Maclaurin's series and it evaluation at $r = 0$ produces

$$E_e = V(0)\int_{\substack{all\\space}} \rho(\vec{r})d^3r + \sum_{u=1}^{3}\frac{\partial V}{\partial u}\int_{\substack{all\\space}} u\rho(\vec{r})d^3r + \frac{1}{2!}\sum_{u=1}^{3}\sum_{v=1}^{3}\frac{\partial^2 V}{\partial u\partial v}\int_{\substack{all\\space}} uv\rho(\vec{r})d^3r + \dots$$

where $u, v \ni \{x, y, x\}$ . The first term is the overall charge distribution $q$ and the second are components of the electric dipole moment of the charge distribution $d$. The third term is related to the nuclear quadrupole, the one of interest here. We can define components of the electric field gradient tensor (at the nucleus origin) as

$$V_{uv} = \frac{\partial^2 V}{\partial u\partial v}$$

These are the second derivatives of the electric potential for the nucleus. The quadrupolar contribution to the electrostatic energy becomes

$$E_e^Q = \frac{1}{2!}\sum_{u=1}^{3}\sum_{v=1}^{3} V_{uv}\int_{\substack{all\\space}} uv\rho(\vec{r})d^3r$$

The potential must satisfy the Laplace equation $\nabla^2 V = 0$, and this translates in the statement that

$$Tr\{V_{uv}\} = \sum_{u=1}^{3} V_{uu} = 0$$

We shall immediately make use of this trace relationship. First, note that we can add virtually any-

[1] C.P. Slichter, "Principles of Magnetic Resonance", 2nd Revised and Expanded Edition, Springer-Verlag New York, 1978. Specifically, see chapter 9.

thing to the classical energy equation, as long as we use and Kronecker delta function and the tensor components.

$$E_e^Q = \frac{1}{2}\sum_{u=1}^{3}\sum_{v=1}^{3}\left[V_{uv}\int_{\substack{all\\space}} uv\rho(\vec{r})d^3r + \delta_{uv}V_{uv}(anything)\right]$$

What we choose to add terms that will place the equation in a form more amenable to tensor nomenclature. This new energy equation is

$$E_e^Q = \frac{1}{2}\sum_{u=1}^{3}\sum_{v=1}^{3}\left[V_{uv}\int_{\substack{all\\space}}\left(uv-\frac{1}{3}\delta_{uv}r^2\right)\rho(\vec{r})d^3r\right]$$

$$E_e^Q = \frac{1}{6}\sum_{u=1}^{3}\sum_{v=1}^{3}\left[V_{uv}\int_{\substack{all\\space}}(3uv-\delta_{uv}r^2)\rho(\vec{r})d^3r\right]$$

We define a new tensor $Q$ where[1]

$$Q_{uv} = \int_{\substack{all\\space}}[3uv-\delta_{uv}r^2]\rho(\vec{r})d^3r$$

and the energy becomes

$$E_e^Q = \frac{1}{6}\sum_{u=1}^{3}\sum_{v=1}^{3}V_{uv}Q_{uv}$$

We can now write the quadrupolar Hamiltonian by replacing the charge distribution (in the nucle-

[1] As will soon be apparent, the form of $Q$ is that of the irreducible tensor component $T_{2,0}$ and this has distinct advantages.

us) $\rho(\vec{r})$ found in the tensor $Q$ as a quantum mechanical operator.

$$\rho(\vec{r}) = \sum_{\vartheta}^{nucleons} q_{\vartheta}\delta(\vec{r}-\vec{r}_{\vartheta}) = \sum_{\vartheta}^{protons} q_{\vartheta}\delta(\vec{r}-\vec{r}_{\vartheta})$$

$$Q_{uv} = \int_{\substack{all\\space}} [3uv - \delta_{uv}r^2]\rho(\vec{r})d^3r = \int_{\substack{all\\space}} [3uv - \delta_{uv}r^2]\left[\sum_{\vartheta}^{protons} q_{\vartheta}\delta(\vec{r}-\vec{r}_{\vartheta})\right]d^3r$$

$$Q_{uv} = e\sum_{\vartheta}^{protons} 3u_{\vartheta}v_{\vartheta} - \delta_{uv}r_{\vartheta}^2$$

$$H^Q = \frac{1}{6}\sum_{u=1}^{3}\sum_{v=1}^{3} V_{uv}Q_{uv}$$

However, the quadrupolar Hamiltonian and energy equations remain in terms of quantities we do not wish to work with in NMR, namely nuclear particles. Our primary task in dealing with the quadrupolar Hamiltonian is to somehow re-write the tensor $Q$ in terms of the overall nucleus (spin). We are only concerned with the nuclear ground states and these are characterized by their total angular momentum $I$, and the associated $2I+1$ angular momentum components. Furthermore we are concerned only with spatial reorientation of nuclei (spin operators remain constant). Thus, it seems evident that we should choose to formulate the elements of $Q$ in terms of the eigenfunctions of angular momentum, that is

$$\langle Im|Q_{uv}|Im'\rangle$$

There is an easy way to relate Cartesian tensor components to angular momentum components, through the Wigner-Ekert theorem. We have

$$\langle Im|Q_{uv}|Im'\rangle = \langle Im|e\sum_{\vartheta}^{protons} 3u_{\vartheta}v_{\vartheta} - \delta_{uv}r_{\vartheta}^2|Im'\rangle = C\langle Im|\left[3\left(\frac{I_uI_v + I_vI_u}{2}\right) - \delta_{uv}I^2\right]|Im'\rangle$$

where $C$ is (as yet) an unknown constant. A particular case would be

$$\langle Im|Q_{zz}|Im\rangle = C\langle Im|\left(\frac{3}{2}(I_zI_z + I_zI_z) - I^2\right)|m\rangle = C\langle Im|(3I_z^2 - I^2)|Im\rangle$$

$$= C[3I^2 - I(I+1)]\langle Im|Im\rangle = CI(2I-1)$$

The quantity $\langle II|Q_{zz}|II\rangle$ is defined to be the quadrupole moment of the nucleus, so we have

$$eQ = CI(2I-1)$$

or equivalently,

$$C = \frac{eQ}{I(2I-1)}$$

Our Cartesian tensor is then given by

$$Q_{uv} = \frac{eQ}{I(2I-1)}\left(\frac{3}{2}(I_uI_v + I_vI_u) - \delta_{uv}I^2\right)$$

We now have the quadrupolar Hamiltonian in a form which can be worked with. Unfortunately the Cartesian tensor $Q_{uv}$ is not in a form which readily translates into irreducible spherical tensor form. Hence, we will need a bit more algebraic manipulation.

$$H^Q = \frac{1}{6}\sum_{u=1}\sum_{v=1} V_{uv}Q_{uv} = \frac{1}{6}\sum_{u=1}\sum_{v=1} V_{uv}\left[\frac{eQ}{I(2I-1)}\left(\frac{3}{2}(I_uI_v + I_vI_u) - \delta_{uv}I^2\right)\right]$$

$$H^Q = \frac{eQ}{6I(2I-1)}\left[\sum^{3}\sum^{3}\frac{3}{2}V_{uv}(I_uI_v + I_vI_u) - \sum^{3} V_{uv}I^2\right]$$

Since the tensor $V$ is traceless, the second term in the previous equation is zero and the Hamiltonian is

$$H^Q = \frac{eQ}{4I(2I-1)}\left[\sum_{u=1}^{3}\sum_{v=1}^{3} V_{uv}(I_uI_v + I_vI_u)\right]$$

Then, if we note that $V$ is symmetric we can write

$$H^Q = \frac{eQ}{2I(2I-1)}\sum_{u=1}^{3}\sum_{v=1}^{3}[V_{uv}I_uI_v]$$

and this is the form which we desire because it can be written as

$$\boldsymbol{H}^Q = \frac{eQ}{2I(2I+1)}\sum_{u}^{axes}\sum_{v}^{axes}\langle 1|\vec{\boldsymbol{I}}|u\rangle\langle u|\hat{\boldsymbol{V}}|v\rangle\langle v|\vec{\boldsymbol{I}}|1\rangle \tag{4-80}$$

or

$$\boldsymbol{H}^Q = \frac{eQ}{2I(2I+1)}\vec{\boldsymbol{I}} \bullet \hat{V} \bullet \vec{\boldsymbol{I}} = \frac{eQ}{2I(2I+1)}\begin{bmatrix}\boldsymbol{I}_x & \boldsymbol{I}_y & \boldsymbol{I}_z\end{bmatrix} \bullet \begin{bmatrix} V_{xx} & V_{xy} & V_{xz} \\ V_{yx} & V_{yy} & V_{yz} \\ V_{zx} & V_{zy} & V_{zz}\end{bmatrix} \bullet \begin{bmatrix}\boldsymbol{I}_x \\ \boldsymbol{I}_y \\ \boldsymbol{I}_z\end{bmatrix} . \tag{4-81}$$

which can be rearranged to produce an equation involving two rank 2 Cartesian tensors by taking the dyadic product of the vector $\vec{\boldsymbol{I}}$ with itself. (We here introduce a spin index.)

$$\boldsymbol{H}_i^Q = \frac{eQ_i}{2I_i(2I_i+1)}\sum_u^{axes}\sum_v^{axes}\langle u|\hat{\boldsymbol{V}}_i|v\rangle\langle v|\vec{\boldsymbol{I}}_i|1\rangle\langle 1|\vec{\boldsymbol{I}}_i|u\rangle = \frac{eQ_i}{2I_i(2I_i+1)}\sum_u^{axes}\sum_v^{axes}\langle u|\hat{\boldsymbol{V}}_i|v\rangle\langle v|\vec{\boldsymbol{I}}_i\vec{\boldsymbol{I}}_i|u\rangle$$

The dyadic product to produce $\vec{\boldsymbol{I}}_i\vec{\boldsymbol{I}}_i$ is explicitly done *via*

$$\begin{bmatrix}\boldsymbol{I}_{ix}\\ \boldsymbol{I}_{iy}\\ \boldsymbol{I}_{iz}\end{bmatrix}\bullet\begin{bmatrix}\boldsymbol{I}_{ix} & \boldsymbol{I}_{iy} & \boldsymbol{I}_{iz}\end{bmatrix} = \begin{bmatrix}\boldsymbol{I}_{ix}\boldsymbol{I}_{ix} & \boldsymbol{I}_{ix}\boldsymbol{I}_{iy} & \boldsymbol{I}_{ix}\boldsymbol{I}_{iz}\\ \boldsymbol{I}_{iy}\boldsymbol{I}_{ix} & \boldsymbol{I}_{iy}\boldsymbol{I}_{iy} & \boldsymbol{I}_{iy}\boldsymbol{I}_{iz}\\ \boldsymbol{I}_{iz}\boldsymbol{I}_{ix} & \boldsymbol{I}_{iz}\boldsymbol{I}_{iy} & \boldsymbol{I}_{iz}\boldsymbol{I}_{iz}\end{bmatrix}.$$

Letting $\hat{T}_i = \vec{\boldsymbol{I}}_i\vec{\boldsymbol{I}}_i$, the quadrupolar Hamiltonian can be expressed as the scalar product of two rank 2 Cartesian tensors.

$$\boldsymbol{H}_i^Q = \frac{eQ_i}{2I_i(2I_i+1)}\sum_u^{axes}\sum_v^{axes}\langle u|\hat{\boldsymbol{V}}_i|v\rangle\langle v|\hat{\boldsymbol{T}}_i|u\rangle = \frac{eQ_i}{2I_i(2I_i+1)}\hat{\boldsymbol{V}}_i\bullet\hat{\boldsymbol{T}}_i \qquad (4\text{-}82)$$

Equation (4-82) can also be rewritten in term of irreducible spherical components rather than in terms of the Cartesian components.

$$\boldsymbol{H}_i^Q = \frac{eQ_i}{2I_i(2I_i+1)}\sum_{l=0}^{2}\sum_m^{\pm l}(-1)^m\boldsymbol{V}_{l-m}^Q(i)\boldsymbol{T}_{lm}^Q(i) \qquad (4\text{-}83)$$

The quadrupolar Hamiltonian at this point is expressed in terms of a product of the nuclear electric quadrupole moment[1] $e\boldsymbol{Q}_i$ and the electric field gradient tensor $\hat{\boldsymbol{V}}$.

We obtain the 9 irreducible spherical components of the quadrupolar spin tensor (rank 2), $T_{l,m}^Q(i)$, directly from the Cartesian components, $\langle v|\hat{\boldsymbol{T}}_i^Q|u\rangle$, as indicated in GAMMA Class Documentation on Spin Tensor. The nomenclature used here for a tensor component is

$$T_{l,m}^Q(i) \quad ,$$

where the subscript *l* spans the rank (in this case 2) as *l* = [0, 2], and the subscript *m* spans *+/- l*, *m* = [-*l*, *l*]. The nine formulas for these quantities a listed in the following figure.

[1] This is a property of the nucleus itself, not of the nucleus environment, and can be found in tables.

### Quadrupolar Irreducible Spherical Spin Tensor Components

$$T^Q_{0,0}(i) = \frac{-1}{\sqrt{3}}\vec{I}_i^2 = \frac{-1}{\sqrt{3}}[I_{ix}^2 + I_{iy}^2 + I_{iz}^2] = \frac{-1}{\sqrt{3}}\left[I_{iz}^2 + \frac{1}{2}(I_{i+}I_{i-} + I_{i-}I_{i+})\right]$$

$$T^Q_{1,0}(i) = \frac{-1}{2\sqrt{2}}(I_{i+}I_{i-} + I_{i-}I_{i+}) \qquad T^Q_{1,\pm1}(i) = \frac{-1}{2\sqrt{2}}[I_{i\pm}\ I_{iz} + I_{iz}I_{i\,\pm}\ \ ]$$

$$T^Q_{2,0}(i) = \frac{1}{\sqrt{6}}[3I_{iz}^2 - \vec{I}_i^2] = \frac{1}{\sqrt{6}}\left[2I_{iz}^2 - \frac{1}{2}(I_{i+}I_{i-} + I_{i-}I_{i+})\right]$$

$$T^Q_{2,\pm1}(i) = \mp\frac{1}{2}[I_{i\pm}\ I_{iz} + I_{iz}I_{i\,\pm}\ \ ] \qquad T^Q_{2,\pm2}(i) = \frac{1}{2}I_{i\,\pm}^2$$

The matrix representation of these nine tensor components will depend upon the matrix representations of the individual spin operators from which they are constructed[1]. These in turn depend upon the spin quantum numbers of the two spins involved. For a treatment of a spin 1 particle the quadrupolar tensor components are expressed in their matrix form in the default product basis of GAMMA as follows[2]. In this case the spin index is implicit.

### Quadrupolar Spherical Spin Tensor Components Matrix Representations

#### Matrix Representations in Single Spin (I=1) Hilbert Space

$$T^Q_{0,0} = \frac{-2}{\sqrt{3}}\begin{bmatrix}1&0&0\\0&1&0\\0&0&1\end{bmatrix} \qquad T^Q_{1,0} = -1\begin{bmatrix}1&0&0\\0&0&0\\0&0&-1\end{bmatrix} \qquad T^Q_{1,1} = \frac{-1}{\sqrt{2}}\begin{bmatrix}0&-1&0\\0&0&-1\\0&0&0\end{bmatrix} \qquad T^Q_{1,-1} = \frac{-1}{\sqrt{2}}\begin{bmatrix}0&0&0\\1&0&0\\0&1&0\end{bmatrix}$$

$$T^Q_{2,0} = \frac{1}{\sqrt{6}}\begin{bmatrix}1&0&0\\0&-2&0\\0&0&1\end{bmatrix} \qquad T^Q_{2,1} = \frac{1}{\sqrt{2}}\begin{bmatrix}0&-1&0\\0&0&1\\0&0&0\end{bmatrix} \qquad T^Q_{2,-1} = \frac{1}{\sqrt{2}}\begin{bmatrix}0&0&0\\1&0&0\\0&-1&0\end{bmatrix} \qquad T^Q_{2,2} = \begin{bmatrix}0&0&1\\0&0&0\\0&0&0\end{bmatrix} \qquad T^Q_{2,-2} = \begin{bmatrix}0&0&0\\0&0&0\\1&0&0\end{bmatrix}$$

The 9 irreducible spherical components of a spatial tensor (rank 2) are formally specified with the nomenclature

$$A_{l,m}(i) \quad ,$$

[1] Note that the spin tensors are invariably constructed in the laboratory coordinate system. Here the z-axis corresponds to the direction of the spectrometer static magnetic field and the coordinate system is right-handed.

[2] The GAMMA program Q_Spin_T.cc was used to generate these matrices. It is listed at the end of this Chapter on page 269.

and are related to its Cartesian components by the following general formulas (See GAMMA Class Documentation on Spatial Tensor).

$$
\begin{gathered}
A_{0,0} = \frac{-1}{\sqrt{3}}[A_{xx} + A_{yy} + A_{zz}] = \frac{-1}{\sqrt{3}}Tr\{A\} \\
A_{1,0} = \frac{-i}{\sqrt{2}}[A_{xy} - A_{yx}] \qquad A_{1,\pm1} = \frac{-1}{2}[A_{zx} - A_{xz} \pm i(A_{zy} - A_{yz})] \\
A_{2,0} = \sqrt{6}[3A_{zz} - (A_{xx} + A_{yy} + A_{zz})] = \sqrt{6}[3A_{zz} - Tr\{A\}] \\
A_{2,\pm1} = \mp\frac{1}{2}[A_{xz} + A_{zx} \pm i(A_{yz} + A_{zy})] \qquad A_{2,\pm2} = \frac{1}{2}[A_{xx} - A_{yy} \pm i(A_{xy} + A_{yx})]
\end{gathered}
\tag{4-84}
$$

Again the subscript *l* spans the rank as *l* = [0, 2], and the subscript *m* spans *+/- l, m* = [-*l*, *l*]. In this quadrupolar treatment we will then have components $V^{Q}_{l,m}(i)$ as indicated in equation (4-46). A general rank two Cartesian tensor can be written as a sum over tensors of ranks 0 - 2 as follows,

Rank 0 Rank 1 Rank 2

$$
\hat{V}_i = \begin{bmatrix} V_{xx} & V_{xy} & V_{xz} \\ V_{yx} & V_{yy} & V_{yz} \\ V_{zx} & V_{zy} & V_{zz} \end{bmatrix}_i = V_{iso}(i)\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} + \begin{bmatrix} 0 & \alpha_{xy} & \alpha_{xz} \\ -\alpha_{xy} & 0 & \alpha_{yz} \\ -\alpha_{xz} & -\alpha_{yz} & 0 \end{bmatrix}_i + \begin{bmatrix} \delta_{xx} & \delta_{xy} & \delta_{xz} \\ \delta_{yx} & \delta_{yy} & \delta_{yz} \\ \delta_{zx} & \delta_{zy} & \delta_{zz} \end{bmatrix}_i
$$

where

$$
V_{iso} = \frac{1}{3}Tr\{\hat{V}\} \qquad \alpha_{xy} = \frac{1}{2}(V_{xy} - V_{yx}) \qquad \delta_{xy} = \frac{1}{2}(V_{xy} + V_{yx} - 2V_{iso})
$$

As the quadrupolar spatial tensor is symmetric and traceless for all spins we obtain a simple result

Rank2

$$
\begin{gathered} V_{iso} = 0 \\ \alpha_{xy} = 0 \\ \delta_{xy} = V_{xy} \end{gathered} \qquad \hat{V}_i = \begin{bmatrix} V_{xx} & V_{xy} & V_{xz} \\ V_{yx} & V_{yy} & V_{yz} \\ V_{zx} & V_{zy} & V_{zz} \end{bmatrix}_i = \begin{bmatrix} \delta_{xx} & \delta_{xy} & \delta_{xz} \\ \delta_{yx} & \delta_{yy} & \delta_{yz} \\ \delta_{zx} & \delta_{zy} & \delta_{zz} \end{bmatrix}_i
$$

This is equivalent to saying that $\hat{V}$ is an irreducible rank 2 tensor. As with any spatial tensor, the quadrupolar spatial tensor can be specified most readily in its principal axis system, the axes set in which the irreducible rank 2 component is diagonal[1].

$$\hat{V}_i(PAS) = \begin{bmatrix} \delta_{xx} & 0 & 0 \\ 0 & \delta_{yy} & 0 \\ 0 & 0 & \delta_{zz} \end{bmatrix}_i$$

These quadrupolar tensors will normally be different for each spin. Rank 2 spatial tensors are commonly specified in their principal axis system by the three components; the isotropic value $A_{iso}$, the anisotropy $\Delta A$, and the asymmetry $\eta$. These are generally given by

$$A_{iso} = \frac{1}{3}Tr\{A\}\ , \qquad \Delta A = A_{zz} - \frac{1}{2}(A_{xx} + A_{yy}) = \frac{3}{2}\delta_{zz} \qquad \eta = (\delta_{xx} - \delta_{yy})/\delta_{zz}$$

A set of Euler angles $\{\alpha, \beta, \gamma\}$ is normally also given to relate the spatial tensor principle axes to another coordinate system. For the quadrupolar spatial tensor we have ($V_{iso} = 0$)

$$\hat{V}_i(PAS) = \delta_{zz}\begin{bmatrix} -\frac{1}{2}(1-\eta) & 0 & 0 \\ 0 & -\frac{1}{2}(1+\eta) & 0 \\ 0 & 0 & 1 \end{bmatrix}_i = eq\begin{bmatrix} -\frac{1}{2}(1-\eta) & 0 & 0 \\ 0 & -\frac{1}{2}(1+\eta) & 0 \\ 0 & 0 & 1 \end{bmatrix}_i \qquad (4\text{-}85)$$

The irreducible spherical elements of the quadrupolar tensor, $V^Q_{2,m}$, in the principal axis system are obtained by placement of (4-35) into (4-51),

$$V^Q_{0,0}(PAS) = 0 \qquad V^Q_{1,m}(PAS) = 0$$

$$V^Q_{2,0}(PAS) = \sqrt{3/2}\delta_{zz} = \sqrt{3/2}eq \qquad V^Q_{2,1}(PAS) = V^Q_{2,-1}(PAS) = 0$$

$$V^Q_{2,2}(PAS) = V^Q_{2,-2}(PAS) = \frac{1}{2}\delta_{zz}\eta = \frac{1}{2}eq\eta$$

Throughout GAMMA, we desire all irreducible spherical rank 2 spatial components to be as similar to normalized spherical harmonics as possible. Thus, we here scale the quadrupolar spatial tensor such that the $2, 0$ component will have a magnitude of the $m = 0$ rank two spherical harmonic when the two spherical angles are set to zero; $Y_{2,0}(\theta, \varphi)\big|_{\theta = \varphi = 0} = \sqrt{5/(4\pi)}$ . We thus define a new quadrupolar spatial tensor such that

$$A^Q_{l,m} = \sqrt{5/(6\pi)}(eq)^{-1}V^Q_{l,m} = \sqrt{5/(6\pi)}\delta^{-1}_{zz}V^Q_{l,m} \qquad (4\text{-}86)$$

[1] The quadrupolar principal axis system for the spin pair has the z-axis pointing along the vector connecting the two spins. The orientation of the x and y axes are inconsequential due to the interaction symmetry about the dipole vector (z-axis).

and rewrite the quadrupolar Hamiltonian given in equation (4-83) as

$$\boldsymbol{H}_i^Q = \frac{eQ_i}{2I_i(2I_i-1)}\left(eq\sqrt{\frac{6\pi}{5}}\right)\sum_{l}^{2}\sum_{m}^{\pm l}(-1)^m A_{l,-m}^Q(i) \bullet \boldsymbol{T}_{l,m}^Q(i)$$

and obtain

$$\boldsymbol{H}_i^Q = \frac{e^2qQ_i}{2I_i(2I_i-1)}\sqrt{\frac{6\pi}{5}}\sum_{l}^{2}\sum_{m}^{\pm l}(-1)^m A_{l,-m}^Q(i) \bullet \boldsymbol{T}_{l,m}^Q(i)$$

The quadrupolar spatial tensors used throughout GAMMA will have the form shown in the following figure (from $A_{l,m}^Q = \sqrt{5/(6\pi)}(eq)^{-1}V_{l,m}^Q$ )

### *Quadrupolar Irreducible Spherical Spatial Tensor Components*

#### *Principal Axis System (PAS)*

$$A_{0,0}^Q(PAS) = 0 \qquad A_{1,0}^Q(PAS) = 0 \qquad A_{1,\pm 1}^Q(PAS) = 0$$

$$A_{2,0}^Q(PAS) = \sqrt{\frac{5}{4\pi}} \qquad A_{2,\pm 1}^Q(PAS) = 0 \qquad A_{2,\pm 2}^Q(PAS) = \sqrt{\frac{5}{24\pi}}\eta$$

We can express the spatial tensor components $A_{l,m}^Q$ relative to any arbitrary axis system (AAS) by a rotation from the principal axes to the new axes *via* the formula

$$A_{2,m}^Q(i, AAS) = \sum_{m'}^{\pm 2} D_{mm'}^2(\Omega) A_{2,m'}^Q(i, PAS) \tag{4-87}$$

where $D_{mm'}^l$ are the rank $l$ Wigner rotation matrix elements and $\Omega$ the set of three Euler angles which relate the principal axes of the chemical shielding tensor to the arbitrary axes. As is evident, regardless of the coordinate system, *only the rank two components will contribute to the quadrupolar Hamiltonian*. Since only the rank 2 components exist we have removed the summation over $l$ and may do the same for the quadrupolar Hamiltonian.

$$\boldsymbol{H}_i^Q = \frac{e^2q_iQ_i}{2I_i(2I_i-1)}\sqrt{\frac{6\pi}{5}}\sum_{m}^{\pm 2}(-1)^m A_{2,-m}^Q(i) \bullet \boldsymbol{T}_{2,m}^Q(i)$$

We simplify our nomenclature by defining a quadrupolar interaction constant as

$$\xi_i^Q = \frac{e^2q_iQ_i}{2I_i(2I_i-1)}\sqrt{\frac{6\pi}{5}} \tag{4-88}$$

The final form of our quadrupolar Hamiltonian for a single spin is then given by

$$\boldsymbol{H}_i^Q = \xi_i^Q \sum_m^{\pm 2} (-1)^m A_{2,-m}^Q(i) \bullet \boldsymbol{T}_{2,m}^Q(i)$$

Note that in the principal axis system this is a relatively simple formula.

$$\boldsymbol{H}_i^Q(PAS) = \xi_i^Q \sum_m^{\pm 2} (-1)^m A_{2,-m}^Q(i, PAS) \bullet \boldsymbol{T}_{2,m}^Q(i)$$

$$\boldsymbol{H}_i^Q(PAS) = \xi_i^Q [A_{2,0}^Q(i, PAS)\boldsymbol{T}_{2,0}^Q(i) + A_{2,2}^Q(i, PAS)\boldsymbol{T}_{2,-2}^Q(i) + A_{2,-2}^Q(i, PAS)\boldsymbol{T}_{2,2}^Q(i)]$$

$$\boldsymbol{H}_i^Q(PAS) = \xi_i^Q \left[ \sqrt{\frac{5}{4\pi}} \left( \frac{1}{\sqrt{6}} [3I_{iz}^2 - \boldsymbol{I}_i^2] \right) + \sqrt{\frac{5}{24\pi}} \eta \left( \frac{1}{2} I_{i-}^2 \right) + \sqrt{\frac{5}{24\pi}} \eta \left( \frac{1}{2} I_{i+}^2 \right) \right]$$

$$\boldsymbol{H}_i^Q(PAS) = \frac{e^2 q Q_i}{4I_i(2I_i - 1)} \left[ 3I_{iz}^2 - \boldsymbol{I}_i^2 + \frac{\eta}{2} I_{i+}^2 + \frac{\eta}{2} I_{i-}^2 \right] \qquad (4\text{-}89)$$

When working with an entire spin system one must sum over all spins with the tensors being in the same coordinate system, for our purposes the laboratory system. The quadrupolar Hamiltonian for a spin system becomes the following.

$$\boldsymbol{H}^Q = \sum_i^{spins} \boldsymbol{H}_i^Q = \sum_i^{spins} \xi_i^Q \sum_m^{\pm 2} (-1)^m A_{2,-m}^Q(i) \bullet \boldsymbol{T}_{2,m}^Q(i) \qquad (4\text{-}90)$$

There is one task remaining, to place our constants into quantities which are easy to work with. A common quantity used in the literature is the quadrupolar coupling constant $QCC$ (shown here in energy units[1]), or in terms of a quadrupolar frequency $\nu^Q$.

$$QCC = e^2 q_i Q_i = 2h\omega^Q = 2h\nu^Q$$

Thus the quadrupolar interaction constant is equivalently expressed as

$$\xi_i^Q = \sqrt{\frac{6\pi}{5}} \frac{e^2 q_i Q_i}{2I_i(2I_i - 1)} = \sqrt{\frac{6\pi}{5}} \frac{QCC_i}{2I_i(2I_i - 1)} = \sqrt{\frac{6\pi}{5}} \frac{2h\nu_i^Q}{2I_i(2I_i - 1)} \qquad (4\text{-}91)$$

Often the quadrupolar coupling constants are often specified in frequency units and this forces the use of planks constant and perhaps factors of $2\pi$ in related equations. How may we best set up the quadrupolar Hamiltonian? We know the form the quadrupolar tensor in terms of the interaction constant and standardized spherical irreducible spatial and spin tensors. We just need to specify the strength and orientation of the individual quadrupolar spatial tensor. In the original un-scaled ten-

[1] In angular frequency units we have $QCC_i = e^2 q_i Q_i / h$ and the units of the coupling constant set the units of both the interaction constant and the Hamiltonian.

sor $\hat{V}$ we had $\delta_{zz}(i) = eq_i$ . For the scaled tensor $\hat{A}$ we will redefine this constant as $\delta_{zz}(i) = QCC_i$ , although we shall always keep it in the interaction constant. It will be used to specify the interaction strength but will not scale the spherical tensor components directly (to insure they behave as normalized spherical harmonics)

The following figures summarize the rank 2 treatment of the quadrupolar Hamiltonian.

## *The Quadrupolar Hamiltonian Summary*

Arbitrary Axis System

$$\boldsymbol{H}_i^Q(AAS) = \xi_i^Q \sum_m^{\pm 2} (-1)^m A_{2-m}^Q(i, AAS)\boldsymbol{T}_{2m}^Q(i, AAS)$$

$$T_{2,0}^Q(i) = \frac{1}{\sqrt{6}}[3I_{iz}^2 - \grave{\boldsymbol{I}}_i^2] = \frac{1}{\sqrt{6}}\left[2I_{iz}^2 - \frac{1}{2}(I_{i+}I_{i-} + I_{i-}I_{i+})\right]$$

$$T_{2,\pm 1}^Q(i) = \mp\frac{1}{2}[I_{i\pm}\ I_{iz} + I_{iz}I_{i\pm}\ ] \qquad T_{2,\pm 2}^Q(i) = \frac{1}{2}I_{i\pm}^2$$

$$\xi_i^Q = \sqrt{\frac{6\pi}{5}}\frac{\delta_{zz}(i)}{2I_i(2I_i - 1)} = \sqrt{\frac{6\pi}{5}}\frac{QCC_i}{2I_i(2I_i - 1)} = \sqrt{\frac{6\pi}{5}}\frac{h\nu_i^Q}{I_i(2I_i - 1)}$$

$$A_{2,0}^Q(PAS) = \sqrt{\frac{5}{4\pi}} \qquad A_{2,\pm 1}^Q(PAS) = 0 \qquad A_{2,\pm 2}^Q(PAS) = \sqrt{\frac{5}{24\pi}}\eta$$

$$A_{2,m}^Q(i, AAS) = \sum_{m'}^{\pm 2} D_{mm'}^2(\varphi_i, \theta_i, \chi_i)A_{2,m'}^Q(i, PAS)\bigg|_{\eta = 0} = Y_{2,m}(\theta_i, \varphi_i)$$

$$T_{2,0}^Q = \frac{1}{\sqrt{6}}\begin{bmatrix} 1 & 0 & 0 \\ 0 & -2 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad T_{2,1}^Q = \frac{1}{\sqrt{2}}\begin{bmatrix} 0 & -1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix} \quad T_{2,-1}^Q = \frac{1}{\sqrt{2}}\begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & -1 & 0 \end{bmatrix} \quad T_{2,2}^Q = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad T_{2,-2}^Q = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix}$$

Principal Axis System

$$\boldsymbol{H}_i^Q(PAS) = \xi_i^Q \sum_m^{\pm 2} (-1)^m A_{2-m}^Q(i, PAS)\boldsymbol{T}_{2m}^Q(i) = \frac{QCC_i}{4I_i(2I_i - 1)}\left[3I_{iz}^2 - \grave{\boldsymbol{I}}_i^2 + \frac{\eta_i}{2}I_{i+}^2 + \frac{\eta_i}{2}I_{i-}^2\right]$$

### 8.7.8 The Random Field Hamiltonian

In this section we consider the Hamiltonian which would result if a spin or spin system is under the influence of random magnetic fields. Often there are species in a liquid NMR sample (solvent molecules, paramagnetic impurities, other solute molecules) which may generate localized magnetic fields which interact with spins in their proximity. Such fields are difficult to quantify but their effects are observable experimentally as they can contribute to relaxation.

We start with the classical interaction energy between a dipole and a magnetic field $\vec{\boldsymbol{B}}_i$ affecting the dipole.

$$E_i = -\vec{\mu}_i \bullet \vec{\boldsymbol{B}}_i$$

where $\vec{\mu}$ is the magnetic moment, $i$ the spin index, $E$ the energy. Here we shall assume that the field is both time dependent and random.

$$\vec{\boldsymbol{B}}_i^{RDM}(t) = \boldsymbol{B}_{i,0}^{RDM}[\boldsymbol{B}_{i,x}^{RDM}(t)\vec{i} + \boldsymbol{B}_{i,y}^{RDM}(t)\vec{j} + \boldsymbol{B}_{i,z}^{RDM}(t)\vec{k}] \tag{4-92}$$

In this formulation, the value $\boldsymbol{B}_{i,0}^{RDM}$ is a time averaged random field magnitude. The associated Hamiltonian is obtained from substitution of $h\gamma\vec{\boldsymbol{I}}_i$ in for $\vec{\mu}_i$ and the random field specified in (4-92) in the classical energy equation.

$$\boldsymbol{H}_i^{RDM} = h\gamma_i\vec{\boldsymbol{I}}_i \bullet \vec{\boldsymbol{B}}_i^{RDM}(t) = h\gamma_i\boldsymbol{B}_{i,0}^{RDM}\sum_{u}^{axes}\langle 1|\vec{\boldsymbol{I}}_i|u\rangle\langle u|\vec{\boldsymbol{B}}_i^{RDM}(t)|1\rangle \tag{4-93}$$

Here, the summation index $u$ spans the three Cartesian axes, $u \in \{x, y, z\}$. In matrix form, this equation looks like

$$\boldsymbol{H}_i^{RDM} = h\gamma_i\boldsymbol{B}_{i,0}^{RDM}\begin{bmatrix}\boldsymbol{I}_{ix} & \boldsymbol{I}_{iy} & \boldsymbol{I}_{iz}\end{bmatrix} \bullet \begin{bmatrix}\boldsymbol{B}_{i,x}^{RDM}(t) \\ \boldsymbol{B}_{i,y}^{RDM}(t) \\ \boldsymbol{B}_{i,z}^{RDM}(t)\end{bmatrix} . \tag{4-94}$$

We shall take a step towards placement of the random field Hamiltonian in the context of a scalar product of two rank 1 Cartesian tensors (vectors) by formally defining spin and spatial tensors $\boldsymbol{T}$ and $\boldsymbol{B}$.

$$\vec{\boldsymbol{T}}_i^{RDM} = \vec{I}_i = \begin{bmatrix} \boldsymbol{I}_{ix} & \boldsymbol{I}_{iy} & \boldsymbol{I}_{iz} \end{bmatrix} \qquad \vec{B}_{i,n}^{RDM}(t) = \begin{bmatrix} \boldsymbol{B}_{i,x}^{RDM}(t) \\ \boldsymbol{B}_{i,y}^{RDM}(t) \\ \boldsymbol{B}_{i,z}^{RDM}(t) \end{bmatrix} \tag{4-95}$$

The vector $\vec{B}_{i,n}^{RDM}(t)$ is scaled from the original $\vec{B}_i^{RDM}(t)$ so that it has a time averaged magnitude of 1. This produces only a trivial variation of equation (4-93)

$$\boldsymbol{H}_i^{RDM} = h\gamma_i \boldsymbol{B}_{i,0}^{RDM} \vec{B}_{i,n}^{RDM}(t) \bullet \vec{\boldsymbol{T}}_i^{RDM}$$

but sets the Hamiltonian into the nomenclature used throughout this Chapter. Rewriting the last equation in terms of irreducible spherical (rank 1) tensor components produces

$$\boldsymbol{H}_i^{RDM} = h\gamma_i \boldsymbol{B}_{i,0}^{RDM} \sum_{l=0}^{1} \sum_{m}^{\pm l} (-1)^m B_{l,-m}^{RDM}(i) \bullet \boldsymbol{T}_{lm}^{RDM}(i) \quad . \tag{4-96}$$

We can obtain the 4 irreducible spherical components of the random field rank 1 spin tensor directly from the Cartesian components, $\langle u|\hat{\boldsymbol{T}}_i^{RDM}|1\rangle$ , as indicated in GAMMA Class Documentation on Spin Tensor. These are

$$\boldsymbol{T}_{l,m}^{RDM}(i) \ ,$$

where $RDM$ signifies the random field interaction and $i$ is the spin index. The tensor index $l$ spans the rank: $l \in [0, 1]$ while the tensor index $m$ spans $l$: $m \in [-l, l]$ The four formulas for these quantities a listed in the following figure.

### *Rank 1 Irreducible Spherical Spin Tensor Components*

$$T_{0,0}(i) = 0 \qquad T_{1,0}(i) = I_{iz} \qquad T_{1,\pm 1}(i) = \frac{\mp 1}{\sqrt{2}} I_{i\pm}$$

For a single spin 1/2 particle the random field spin tensor components are, in the Hilbert space of the spin itself, given in the following figure (the spin index is implicit here)

### Random Field Rank 1 Irreducible Spherical Spin Tensor Components Matrix Representations in 1-spin (I=1/2) Hilbert Space

$$T_{0,0}^{(1)} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \qquad T_{1,0}^{(1)} = \frac{1}{2}\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \qquad T_{1,-1}^{(1)} = \frac{1}{\sqrt{2}}\begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix} \qquad T_{1,1}^{(1)} = \frac{-1}{\sqrt{2}}\begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}$$

The 4 irreducible spherical components of the random field spatial tensor (rank 1) are formally specified with the nomenclature

$$B_{l,m}^{RDM}(i,t) \quad ,$$

where again the subscript *l* spans the rank as *l* = [0, 1], and the subscript *m* spans +/- *l*, *m* = [-*l*, *l*]. The four formulas which produce these quantities are

### Un-scaled Random Field Irreducible Spherical Spatial Tensor Components

$$B_{0,0}^{RDM}(i,t) = 0$$

$$B_{1,0}^{RDM}(i,t) = B_{i,z}^{RDM}(t) \qquad B_{1,\pm1}^{RDM}(i,t) = \mp\sqrt{\frac{1}{2}}B_{i,\pm}^{RDM}(t) = \mp\sqrt{\frac{1}{2}}[B_{i,x}^{RDM}(t) \pm iB_{i,y}^{RDM}(t)]$$

In keeping with the other Hamiltonians presented in this Chapter, we can scale our spatial tensor components so that they relate directly to spherical harmonics, in this instance rank 1 spherical harmonics[1].

$$Y_{1,0} = \sqrt{\frac{3}{4\pi}}\cos\theta = \sqrt{\frac{3}{4\pi}}\frac{z}{r} \qquad Y_{1,\pm1} = \mp\sqrt{\frac{3}{8\pi}}\sin\theta e^{\pm i\phi} = \mp\sqrt{\frac{3}{8\pi}}\frac{(x \pm iy)}{r} \qquad (4\text{-}97)$$

In order to do so, we may think of the instantaneous field magnitude as the radius "$r$" and the respective field components as $x$, $y$, and $z$. The magnitude of the random field at any one time is given by (only the time average is unity)

$$\boldsymbol{B}_{i,n}^{RDM}(t) = \left|\vec{\boldsymbol{B}}_{i,n}^{RDM}(t)\right| = \sqrt{[\boldsymbol{B}_{i,x}^{RDM}(t)]^2 + [\boldsymbol{B}_{i,y}^{RDM}(t)]^2 + [\boldsymbol{B}_{i,z}^{RDM}(t)]^2} \qquad (4\text{-}98)$$

[1] Whereas the time averaged Cartesian components were set to 1, the normalized spherical harmonics have a spatial integration of 1. This was of use in obtaining spectral density functions but does not apply to the random field mechanism because there is no way to quantitate the motions involved.

Thus, the irreducible spherical components of the spatial tensor can be written as

$$B_{1,0}^{RDM}(i,t) = B_{i,z}^{RDM}(t) = \sqrt{\frac{4\pi}{3}}\boldsymbol{B}_{i,n}^{RDM}(t)\left[\sqrt{\frac{3}{4\pi}}\frac{B_{i,z}^{RDM}(t)}{\boldsymbol{B}_{i}^{RDM}(t)}\right] = \sqrt{\frac{4\pi}{3}}\boldsymbol{B}_{i,n}^{RDM}(t)Y_{1,0}(i,t)$$

$$B_{1,\pm1}^{RDM}(i,t) = \mp\sqrt{\frac{1}{2}}B_{i,\pm}^{RDM}(t) = \sqrt{\frac{4\pi}{3}}\boldsymbol{B}_{i,n}^{RDM}(t)\left[\mp\sqrt{\frac{3}{8\pi}}\frac{B_{i,\pm}^{RDM}(t)}{\boldsymbol{B}_{i}^{RDM}(t)}\right] = \sqrt{\frac{4\pi}{3}}\boldsymbol{B}_{i,n}^{RDM}(t)Y_{1,\pm1}(i,t)$$

We shall now define a new spatial tensor, $A^{RDM}$ whose irreducible spherical components relate to the those of the spatial tensor of the field as

$$A_{lm}^{RDM}(i,t) = \sqrt{\frac{3}{4\pi}}\frac{B_{lm}^{RDM}(i,t)}{\boldsymbol{B}_{i,n}^{RDM}(t)} = Y_{lm}(i,t) \qquad (4\text{-}99)$$

These tensor components look particularly simple,

***Rank 1 Random Field Irreducible Spherical Spatial Tensor Components***

$$A_{0,0}^{RDM}(i,t) = 0 \qquad A_{1,0}^{RDM}(i,t) = Y_{1,0}(i,t) \qquad A_{1,\pm1}^{RDM}(i,t) = Y_{1,\pm1}(i,t)$$

but they turn out to be of no more use than the un-scaled field components in the spectral density function formulation.

Defining the random field interaction constant as[1]

$$\xi_i^{RDM} = h\gamma_i\boldsymbol{B}_{i,0}^{RDM}\sqrt{\frac{4\pi}{3}}\,, \qquad (4\text{-}100)$$

we now rewrite the random field Hamiltonian with the scaled spatial tensor and interaction constant. From (4-96) by substitution of equations (4-99) and (4-100),

$$\boldsymbol{H}_i^{RDM}(t) = \xi_i^{RDM}\boldsymbol{B}_{i,n}^{RDM}(t)\sum_m^{\pm1}(-1)^m\boldsymbol{A}_{1,-m}^{RDM}(i,t)\bullet\boldsymbol{T}_{1m}^{RDM}(i) \qquad (4\text{-}101)$$

Notice that this does not quite fit our generalized Hamiltonian form, and as a result the formulation of the random field spectral density functions will be different (than those which do fit the generalized form).

[1] Note that since random field interactions strengths are empirically determined, the interaction constants will typically not be explicitly determined.

We now have the Hamiltonian represented by normalized spatial fluctuations, $\boldsymbol{A}_{1,-m}^{RDM}$ , in the sense that they will integrate to unity over 3 dimensional space. Yet these components are randomly fluctuating in orientation, thus average in time to zero. In turn, they are multiplied by another function which is also randomly fluctuating in magnitude, $\boldsymbol{B}_{i,n}^{RDM}(t)$, but it's average over time is not zero, it is 1.

By summing over all the spins in a spin system we attain the total random field Hamiltonian.

$$\boldsymbol{H}^{RDM}(t) = \sum_{i}^{spins} \xi_i^{RDM}\boldsymbol{B}_{i,n}^{RDM}(t)\sum_{m}^{\pm 1}(-1)^m\boldsymbol{A}_{1,-m}^{RDM}(i,t)\bullet\boldsymbol{T}_{1m}^{RDM}(i) \qquad (4\text{-}102)$$

If the agent(s) responsible for the random field are assumed to have no particular orientation, the generated field is isotropic. This means that the time averaged field amplitude is the same in all directions.

$$\sqrt{\langle[\boldsymbol{B}_{i,x}^{RDM}(t)]^2\rangle} = \sqrt{\langle[\boldsymbol{B}_{i,y}^{RDM}(t)]^2\rangle} = \sqrt{\langle[\boldsymbol{B}_{i,z}^{RDM}(t)]^2\rangle} = \frac{1}{3}\sqrt{\langle\vec{\boldsymbol{B}}_{i,n}^{RDM}(t)|\vec{\boldsymbol{B}}_{i,n}^{RDM}(t)\rangle}$$

We have already defined the time averaged value of the vector $\vec{\boldsymbol{B}}_{i,n}^{RDM}(t)$ to be unity

$$\sqrt{\langle\vec{\boldsymbol{B}}_{i,n}^{RDM}(t)|\vec{\boldsymbol{B}}_{i,n}^{RDM}(t)\rangle} = 1$$

It makes no sense to think of this Hamiltonian in a particular frame (relative to any specific set of axes), so we have not dealt with any rotations on $\boldsymbol{H}^{RDM}$. Because of its random nature, in a liquid system we should only see its time-averaged effects, the manifestation of which will be relaxation. It will not change any of the static energy levels.

We now regroup all the applicable equations for dealing with the random field Hamiltonian.

## The Random Field Hamiltonian Summary

$$\boldsymbol{H}^{RDM}(t) = \sum_{i}^{spins} \boldsymbol{H}_i^{RDM}(t)$$

$$\boldsymbol{H}_i^{RDM}(t) = \xi_i^{RDM}\boldsymbol{B}_{i,n}^{RDM}(t)\sum_{l=0}^{1}\sum_{m}^{\pm l}(-1)^m A_{l-m}^{RDM}(i,t)\boldsymbol{T}_{lm}^{RDM}(i)$$

$$\xi_i^{RDM} = h\gamma_i\sqrt{\frac{4\pi}{3}}\boldsymbol{B}_{i,0}^{RDM}$$

$$A_{0,0}^{RDM}(i,t) = 0 \qquad A_{1,0}^{RDM}(i,t) = Y_{1,0}(i,t) \qquad A_{1,\pm1}^{RDM}(i,t) = Y_{1,\pm1}(i,t)$$

$$T_{0,0}^{RDM}(i) = 0 \qquad T_{1,0}^{RDM}(i) = I_{iz} \qquad T_{1,\pm1}^{RDM}(i) = \frac{\mp1}{\sqrt{2}}I_{i\pm}$$

$$Y_{1,m}(i,t) = \sqrt{\frac{3}{4\pi}}\frac{\boldsymbol{B}_{1,m}^{RDM}(i,t)}{\boldsymbol{B}_{i,n}^{RDM}(t)}$$

$$\boldsymbol{B}_{0,0}^{RDM}(i,t) = 0 \qquad \boldsymbol{B}_{1,0}^{RDM}(i,t) = B_{i,z}^{RDM}(t) \qquad \boldsymbol{B}_{1,\pm1}^{RDM}(i,t) = \mp\sqrt{\frac{1}{2}}B_{i,\pm}^{RDM}(t)$$

$$\boldsymbol{B}_{i,n}^{RDM}(t) = \left|\vec{\boldsymbol{B}}_i^{RDM}(t)\right| = \sqrt{[\boldsymbol{B}_{i,x}^{RDM}(t)]^2 + [\boldsymbol{B}_{i,y}^{RDM}(t)]^2 + [\boldsymbol{B}_{i,z}^{RDM}(t)]^2}$$

$$\vec{\boldsymbol{B}}_i^{RDM}(t) = \boldsymbol{B}_{i,0}^{RDM}[\boldsymbol{B}_{i,x}^{RDM}(t)\vec{i} + \boldsymbol{B}_{i,y}^{RDM}(t)\vec{j} + \boldsymbol{B}_{i,z}^{RDM}(t)\vec{k}]$$

$$\sqrt{\langle[\boldsymbol{B}_{i,x}^{RDM}(t)]^2\rangle} = \sqrt{\langle[\boldsymbol{B}_{i,y}^{RDM}(t)]^2\rangle} = \sqrt{\langle[\boldsymbol{B}_{i,z}^{RDM}(t)]^2\rangle} = \frac{1}{3}\sqrt{\langle\vec{\boldsymbol{B}}_{i,n}^{RDM}(t)|\vec{\boldsymbol{B}}_{i,n}^{RDM}(t)\rangle} = \frac{1}{3}$$

## 8.7.9 The Chemical Shift Hamiltonian

The chemical shift Hamiltonian combines the isotropic parts of the Zeeman Hamiltonian and the Chemical Shielding Hamiltonian then references these contributions to specific frequencies. In frequency units this is then

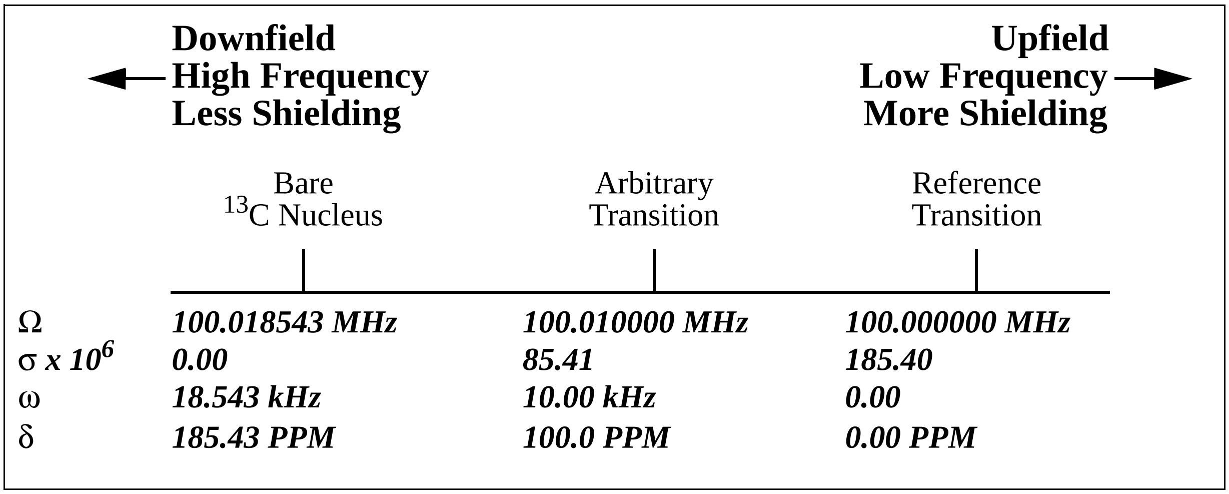
$$\boldsymbol{H}_{cs} = H^{Z} + H^{CSI} + \sum_{i}^{spins} \Omega_{i,\,ref} \boldsymbol{I}_{zi}\,, \tag{4-103}$$

or equivalently

$$\boldsymbol{H}_{cs} = \sum_{i}^{spins} -\gamma_i B_o [1-\sigma_{iso}(i)]\boldsymbol{I}_z + \Omega_{i,\,ref}\boldsymbol{I}_{zi} = \sum_{i}^{spins} (\Omega_{i,\,o} - \Omega_{i,\,ref})\boldsymbol{I}_{zi} \tag{4-104}$$

The reader is here reminded of the overall frequency scales involved in NMR. The small contribution from a positive shielding opposes the large contribution from a positive Zeeman term.

**_Comparison of Shielding vs. Reported Shifts in a $^{13}$C Spectrum_**



| | Bare $^{13}$C Nucleus | Arbitrary Transition | Reference Transition |
|---|---|---|---|
| Ω | *100.018543 MHz* | *100.010000 MHz* | *100.000000 MHz* |
| σ x $10^6$ | *0.00* | *85.41* | *185.40* |
| ω | *18.543 kHz* | *10.00 kHz* | *0.00* |
| δ | *185.43 PPM* | *100.0 PPM* | *0.00 PPM* |

*Figure 19-14* : Comparison of different shifts in a $^{13}$C spectrum. The static magnetic field is 9.4 Tesla ($^{1}$H = 400 MHz, $^{13}$C =100.02 MHz) and TMS is used for the standard reference (185.4 PPM).

Thus the chemical shift Hamiltonian contains only difference frequencies.

$$\boldsymbol{H}_{cs} = \sum_{i}^{spins} -h\omega_i \boldsymbol{I}_z\,, \tag{4-105}$$

There are a few points worthy of mention here. First, unless the user explicitly switches rotating frames, the reference frequency/frequencies is/are used for the rotating frame. This is equivalent in experimental terms to setting the spectrometer frequency equal to the offset frequency. The shift frequencies input when constructing a chemical shift Hamiltonian are then relative both the offset and the spectrometer frequency. Secondly, for a positive chemical shift the precession direction will be the same a that for the Zeeman contribution. Third, chemical shifts are often input in PPM units rather than Hz. They relate via the relationship

$$\delta(PPM) \;=\; \frac{\Omega - \Omega_{ref}}{\Omega_{ref}} \times 10^6 \;=\; \frac{\omega}{\Omega_{ref}} \times 10^6 \qquad (4\text{-}106)$$

Finally, due to the way this Hamiltonian is constructed, it applies to the Liouville equation in the rotating frame or multiple rotating frames.

### 8.7.10 The High Resolution NMR Hamiltonian

For spin systems which are rapidly reorienting. The “high resolution” Hamiltonian applied to liquid systems in NMR is just this, an average Hamiltonian.

Keeping only the rank 0, rotationally independent components of the Hamiltonians present

$$\boldsymbol{H}_0 \;=\; \sum_{i}^{spins} \boldsymbol{H}_0(i) \;=\; \sum_{i}^{spins} \{\boldsymbol{H}_i^Z + \boldsymbol{H}_i^D + \boldsymbol{H}_i^{CS} + \boldsymbol{H}_i^J + \boldsymbol{H}_i^Q + \boldsymbol{H}_i^R\} \quad , \qquad (4\text{-}107)$$

$$\boldsymbol{H}_0 \;=\; \sum_{i}^{spins} \{\boldsymbol{H}_i^Z + \boldsymbol{H}_i^D + \boldsymbol{H}_i^{CSI} + \boldsymbol{H}_i^J + \boldsymbol{H}_i^Q + \boldsymbol{H}_i^R\} \;, \qquad (4\text{-}108)$$

$$\boldsymbol{H}_0 \;=\; \sum_{i}^{spins} \left| \left\{ -h\gamma_i B_o \vec{\boldsymbol{I}}_i \bullet \vec{\boldsymbol{B}}_n + 0 + h\gamma_i B_o \sigma_{iso}(i) \vec{\boldsymbol{I}}_i \bullet \vec{\boldsymbol{B}}_n + \sum_{j>i}^{spins} hJ_{iso}(i,j) \vec{\boldsymbol{I}}_i \bullet \vec{\boldsymbol{I}}_j + 0 + 0 \right\} \right| \qquad (4\text{-}109)$$

$$\boldsymbol{H}_0 \;=\; \sum_{i}^{spins} \left| \left\{ -h\gamma_i B_o [1 - \sigma_{iso}(i)] \vec{\boldsymbol{I}}_i \bullet \vec{\boldsymbol{B}}_n + \sum_{j>i}^{spins} hJ_{iso}(i,j) \vec{\boldsymbol{I}}_i \bullet \vec{\boldsymbol{I}}_j \right\} \right| \quad , \qquad (4\text{-}110)$$

The form of this Hamiltonian which is commonly found in the literature is obtained by substitution of the shielding frequency, $\vec{\boldsymbol{B}}_o$

$$\boldsymbol{H}_0 \;=\; \sum_{i}^{spins} -h\omega_i \vec{\boldsymbol{I}}_z + \sum_{i}^{spins} \sum_{j>i}^{spins} hJ_{iso}(i,j) \vec{\boldsymbol{I}}_i \bullet \vec{\boldsymbol{I}}_j \quad , \qquad (4\text{-}111)$$

accounts for the interaction between magnetic moment (a nuclear spin) and an applied static magnetic field $\vec{\boldsymbol{B}}_o$. The classical interaction energy between an applied field and a nuclear spin is

## 8.7.11 The RF Field Hamiltonian

We now consider an applied rf-field of angular frequency $\Omega_{rf}$ and phase angle $\phi$ applied along the x-axis in the laboratory frame. Its oscillating magnetic field can be represented by[1]

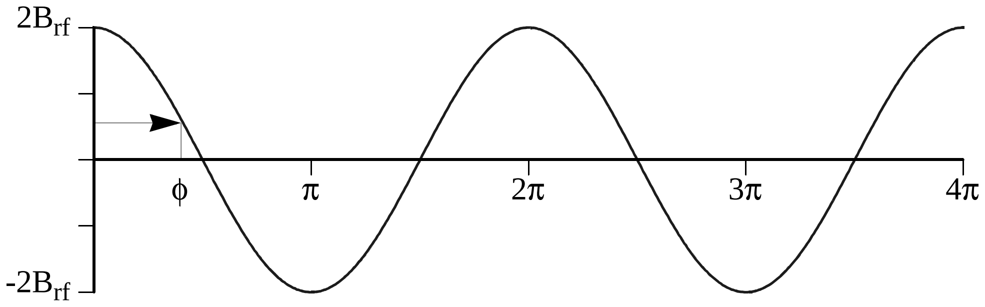
$$\vec{\boldsymbol{B}}_{rf}(t) = 2\left|\vec{\boldsymbol{B}}_{rf}\right| \cos(\Omega_{rf}t - \phi)\hat{\boldsymbol{i}} \quad . \tag{4-112}$$

Since alternate definitions abound in the literature we shall take extreme caution in setting up our Hamiltonian. In the GAMMA definition, the phase angle is a distance along the +x axis in degrees as easily seen from both the Table below and a plot of $\vec{\boldsymbol{B}}_{rf}(t)$ versus $\Omega_{rf}t$.

**Table 3: Values of $X = \cos(\Omega_{rf}t - \phi)$ and the Behavior of $\vec{\boldsymbol{B}}_{rf}(t)$**

| | RF-Field Phase Angle | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| $\Omega_{rf}t$ | $\phi = 0$ | | $\phi = 90$ | | $\phi = 180$ | | $\phi = 270$ | |
| | $\Omega_{rf}t - \phi$ | X | $\Omega_{rf}t - \phi$ | X | $\Omega_{rf}t - \phi$ | X | $\Omega_{rf}t - \phi$ | X |
| 0 | 0 | 1 | -90 | 0 | -180 | -1 | -270 | 0 |
| 90 | 90 | 0 | 0 | 1 | -90 | 0 | -180 | -1 |
| 180 | 180 | -1 | 90 | 0 | 0 | 1 | -90 | 0 |
| 270 | 270 | 0 | 180 | -1 | 90 | 0 | 0 | 1 |
| Function | Cosine | | Sine | | -Cosine | | -Sine | |

**$\vec{\boldsymbol{B}}_{rf}(t)$ versus $\Omega_{rf}t$ depicting the phase angle $\phi$**



[1] The factor of 2 in this equation if merely for convenience. It will disappear from successive equations as they are manipulated algebraically. The phase angle $\phi$ allows for pulses about any axis in the xy-plane and maintains complete generality of the equation.

As is commonly done in the literature[1], $\boldsymbol{B}_{rf}(t)$ (and its associated Hamiltonian $\boldsymbol{H}^{rf}(t)$ ) can be broken up into two components having equal magnitudes and angular frequencies but rotating in opposite directions in the xy-plane.

$$\vec{\boldsymbol{B}}_{rf}(t) = 2\left|\vec{\boldsymbol{B}}_{rf}\right|\cos(\Omega_{rf}t-\phi)\hat{\boldsymbol{i}}$$

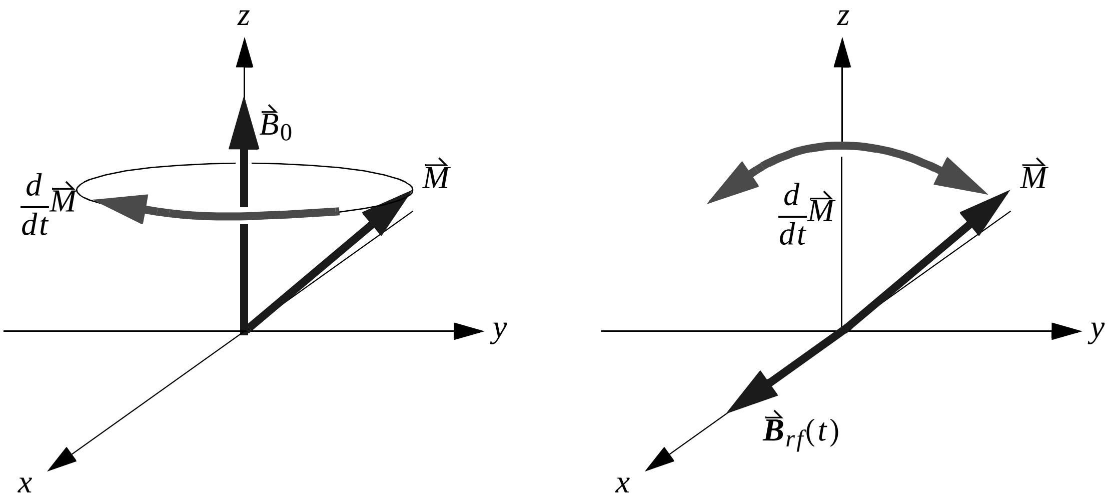
$$\vec{\boldsymbol{B}}_{rf}(t) = \boldsymbol{B}_{rf}[\underset{\text{counter-clockwise}}{[\cos(\Omega_{rf}t-\phi)\boldsymbol{i}+\sin(\Omega_{rf}t-\phi)\boldsymbol{j}]} + \underset{\text{clockwise}}{[\cos(\Omega_{rf}t-\phi)\boldsymbol{i}-\sin(\Omega_{rf}t-\phi)\boldsymbol{j}]}]$$

Before continuing we shall attempt to gain a bit of physical intuition as to how this field affects spins. From a classical standpoint we know that a magnetization $\vec{M}$ interacting with a magnetic field $\vec{B}$ will experience a torque proportional to

$$\frac{d}{dt}\vec{M} \propto \vec{M}\times\vec{B}$$

Thus, for magnetization under the influence of the static field $\vec{B}_0$ along the +z axis we conclude using the right hand rule that any magnetization should precess in a counter-clockwise fashion about the z-axis as shown in the figure on the left.

### Classical Magnetization Response to Magnetic Fields



As for magnetization under the influence of the applied field $\vec{\boldsymbol{B}}_{rf}(t)$ , according to equation (4-112) we expect that the direction of torque will be about the x-axis but changing in direction at a rate which depends upon the field frequency. However, in NMR we normally never deal with a system under the influence of an applied field without the simultaneous presence of the much stronger static field. Thus, the actual magnetization response will be a combination of the two figures. This is difficult visualize unless we think in terms of a rotating frame which moves counter-clockwise with the motion due to $\vec{\boldsymbol{B}}_0$ and think of $\vec{\boldsymbol{B}}_{rf}(t)$ in terms of the two components which are also oscil-

[1] The reason for doing so is to ultimately switch into a rotating frame in which both the field and its Hamiltonian appear time independent. For example, see **********

lating. If we rotate with $\vec{M}$, say at a frequency $\Omega_{rot}$ we will see that the counter-clockwise component of $\vec{\boldsymbol{B}}_{rf}(t)$ is oscillating in the xy-plane at a frequency $\Omega_{rf}$ less the frequency we are rotating at $\Omega_{rot}$. On the other hand, the clockwise component of $\vec{\boldsymbol{B}}_{rf}(t)$ will be appear to be rotating at $\Omega_{rf} + \Omega_{rot}$. The torque on $\vec{M}$ due to the field will still appear chaotic, essentially its effects rapidly cancelling out, unless $\Omega_{rf}$ matches, or nearly matches, $\Omega_{rot}$. By choosing a proper applied field frequency, the counter-clockwise component in the rotating frame appears static and will effectively place a torque on the magnetization. The clockwise component will have essentially not effect, the torque on the magnetization from this component is rapidly changing direction in time and averages to nothing.

We arrive at the following conclusion based on classical arguments. The applied rf-field will have little to no effect unless a field frequency is chosen which closely matches the Larmor precessional frequency of some spin(s). Furthermore, it can only the counter-clockwise component of the field which has any (if any) effect. Then to a high degree of approximation,

$$\vec{\boldsymbol{B}}_{rf}(t) = 2|\vec{\boldsymbol{B}}_{rf}|\cos(\Omega_{rf}t - \phi)\hat{\boldsymbol{i}} = \boldsymbol{B}_{rf}[\cos(\Omega_{rf}t - \phi)\boldsymbol{i} + \sin(\Omega_{rf}t - \phi)\boldsymbol{j}] \tag{4-113}$$

where we have thrown out the component which oscillations in the direction opposite any Larmor precession due to $\vec{\boldsymbol{B}}_0$.

We now return to quantum mechanical aspects of the treatment. Since, in general, the classical energy of a magnetic dipole interacting with a magnetic field is

$$\boldsymbol{E} = -\vec{\mu} \bullet \vec{\boldsymbol{B}} ,$$

the energy of a dipole moment interacting with the rf-field under consideration is simply

$$\boldsymbol{E}^{rf}(t) = -\vec{\mu} \bullet \vec{\boldsymbol{B}}_{rf}(t) \quad .$$

Replacing $\vec{\mu}$ with $\gamma h \vec{\boldsymbol{I}}$ produces the Hamiltonian for the interaction. For a single spin $i$ and the oscillating field given by (4-112) this is, using now $\boldsymbol{B}_{rf} = |\vec{\boldsymbol{B}}_{rf}|$

$$\boldsymbol{H}_i^{rf}(t) = -h\gamma_i \boldsymbol{B}_{rf} \vec{\boldsymbol{I}}_i \bullet [\cos(\Omega_{rf}t - \phi)\boldsymbol{i} - \sin(\Omega_{rf}t - \phi)\boldsymbol{j}]$$

$$\boldsymbol{H}_i^{rf}(t) = -h\gamma_i \boldsymbol{B}_{rf}[\cos(\Omega_{rf}t - \phi)\boldsymbol{I}_{ix} - \sin(\Omega_{rf}t - \phi)\boldsymbol{I}_{iy}]$$

This is simply the scaled $\boldsymbol{I}_{ix}$ vector rotating about the z-axis and can in fact be rewritten in terms of rotations about z.

$$\boldsymbol{H}_i^{rf}(t) = -(h\gamma_i)\boldsymbol{B}_{rf}\boldsymbol{R}_z^{-1}(\Omega_{rf}t)\boldsymbol{R}_z(\phi)\boldsymbol{I}_{ix}\boldsymbol{R}_z^{-1}(\phi)\boldsymbol{R}_z(\Omega_{rf}t)$$

The total rf-field Hamiltonian for a spin system is obtained by summing over all spins,

$$\boldsymbol{H}_i^{rf}(t) = -h\boldsymbol{B}_{rf}\sum_{i}^{spins}\{\gamma_i \boldsymbol{R}_z^{-1}(\Omega_{rf}t)\boldsymbol{R}_z(\phi)\boldsymbol{I}_{ix}\boldsymbol{R}_z^{-1}(\phi)\boldsymbol{R}_z(\Omega_{rf}t)\}$$

For convenience we can define the total magnetic moment operator as

$$\vec{M} = \sum_{i}^{spins} \gamma_i \vec{\boldsymbol{I}}_i$$

and the phased x-component of this operator as

$$M_{x,\varphi} = \boldsymbol{R}_z(\phi) M_x \boldsymbol{R}_z^{-1}(\phi) \quad .$$

Our total rf-field Hamiltonian is then

$$\boldsymbol{H}_i^{rf}(t) = -h\boldsymbol{B}_{rf}\{\boldsymbol{R}_z^{-1}(\Omega_{rf}t) M_{x,\varphi} \boldsymbol{R}_z(\Omega_{rf}t)\} \qquad (4\text{-}114)$$

# 8.8 Example Source Codes

## HShift.cc

The Chemical Shift Hamiltonian

```
/* HShift.cc ********************************************-*-c++-*-
**                                                              **
**             Example program for the GAMMA Library            **
**                                                              **
** This program looks at the chemical shift Hamiltonian that is **
** provided by the function Hcs in the NMR Library module       **
** nmr_ham. Hcs is the isotropic part of the chemical sheilding **
** Hamiltonian. It describes the interaction of a magnetic      **
** moment with the localized magnetic field generated from the  **
** surrounding electron clouds response to the magnetic field   **
** of the spectrometer.                                         **
**                                                              **
** In this example, a single spin system is constructed and     **
** the spin assigned a positive chemical shift value. Then      **
** the system is pulsed so that the bulk magnetizaton is no     **
** longer aligned with the field. Then, the precession of the   **
** magnetization is followed in time in the rotating frame at   **
** the Larmor frequency of the spin’s isotope type. Thus the    **
** precession takes place at the sheilding frequency.           **
**                                                              **
** According to classical physics, the magnetization should     **
** experience a torque given by (Bo is along +z in GAMMA)       **
**                     ->                                       **
**                     dM   ->    ->                            **
**                     -- = M  x  B                             **
**                     dt          o,z                          **
**                                                              **
** and thus evolve about the z-axis in a clockwise direction    **
** when looking down into the xy-plane from the +z-axis. As     **
** the spin is sheilded, its precessional frequency will be     **
** less than the Larmor frequency of the isotope type by the    **
** shift value. Furthermore, since we will be in a rotating     **
** frame at this Larmor frequency, we do not see the clockwise  **
** precession which exists in the lab frame (in Megahertz) but  **
** a counter-clockwise precession in Hz!!!                      **
**                                                              **
*****************************************************************/

#include <gamma.h>
main (int argc, char* argv[])
  {
```

```
//                       Output a Header
 cout << “\n\n Single Spin Magnetization Evolution Under Shift Ham.\n\n”;

//                       Setup

 spin_system sys(1);                    // Set up a single spin system
 sys.Omega(500.0);                      // Set field at 500 MHz
 sys.shift(0, 200.0);                   // Set the shift to be 200 Hz
 gen_op H = Hcs(sys);                   // Get isotropic shift Ham.
 double Mx, My, Mz;                     // Single M components
 gen_op FX = gen_op(Fx(sys), sys.get_basis());
 gen_op FY = gen_op(Fy(sys), sys.get_basis());
 gen_op FZ = gen_op(Fz(sys), sys.get_basis());

 gen_op sigmaeq = sigma_eq(sys);        // Density matrix for equilibrium
 Mx = Re(trace(FX, sigmaeq));           // Get x-magnetization component
 My = Re(trace(FY, sigmaeq));           // Get y-magnetization component
 Mz = Re(trace(FZ, sigmaeq));           // Get z-magnetization component
 cout << “\n\nEquilibruim Magnetization: “
      << Mx << “\t” << My << “\t” << Mz;
 gen_op sigma = Ixpuls(sys,sigmaeq,-45);// Apply an (PI/2)x pulse
 Mx = Re(trace(FX, sigma));             // Get x-magnetization component
 My = Re(trace(FY, sigma));             // Get y-magnetization component
 Mz = Re(trace(FZ, sigma));             // Get z-magnetization component
 cout << “\n\nMagnetization After Pulse: “
      << Mx << “\t” << My << “\t” << Mz;

 int npts = 501;                        // Number of magnetization points
 block_1D Mdata(npts);                  // For storing M x & y components
 double ttot = 2.e-2;                   // Total time for 4 cycles
 double tinc = ttot/double(npts-1);     // Time increment between points

//      Begin Looping Over Different Times

 double time;
 gen_op sigmat;
 for(int j=0; j<npts; j++)
   {
   time = double(j)*tinc;               // Total applied field length
   sigmat = evolve(sigma, H, time);     // Evolve the Shift Hamiltonian
   Mx = Re(trace(FX, sigmat));          // Get x-magnetization component
   My = Re(trace(FY, sigmat));          // Get y-magnetization component
   Mz = Re(trace(FZ, sigmat));          // Get z-magnetization component
   Mdata(j) = complex(Mx,My);           // Store x & y magnetization
   cout << “\n”<< Mx << “\t”
        << My << “\t” << Mz;
   }
 FM_1D(“Shift.mif”,Mdata,14,5,0,2,2);   // Output mag. plot, FM format
```

```
       cout << “\n”;                            // End with screen nice
       }
```

## HZeeman.cc

### *The Zeeman Hamiltonian*

```
/* HZeeman.cc *******************************************-*-c++-*-
**                                                              **
** Example program for the GAMMA Library                        **
**                                                              **
** This program looks at the Zeeman Hamiltonian provided by     **
** the function Hz in the NMR Library module nmr_ham. Hz        **
** is the Hamiltonian describing the interaction of a magnetic  **
** moment with the static magnetic field of the spectrometer.   **
**                                                              **
** In this example, a single spin system is constructed and     **
** pulsed so that the bulk magnetizaton is no-longer aligned    **
** with the field. Then, the precession of the magnetization    **
** is followed in time in the laboratory frame, i.e. as the     **
** precession takes place in MHz.                               **
**                                                              **
** According to classical physics, the magnetization should     **
** experience a torque given by(B is along +z in GAMMA)         **
**                               o                              **
**                      ->                                      **
**                      dM   ->    ->                           **
**                      -- = M  x  B                            **
**                      dt          o,z                         **
**                                                              **
** and thus evolve about the z-axis in a clockwise direction    **
** when looking down into the xy-plane from the +z-axis.        **
**                                                              **
******************************************************************/

#include <gamma.h>

main (int argc, char* argv[])

 {
//                       Output a Header

 cout << “\n\n\t\t\tMagnetization Evolution Under Zeeman Ham.\n\n”;

//                       Setup

 spin_system sys(1);                    // Set up a single spin system
 sys.Omega(500.0);                      // Set field at 500 MHz
 gen_op H = Hz(sys);                    // Get the Zeeman Hamiltonian
 double Mx, My, Mz;                     // Single M components
 gen_op FX = gen_op(Fx(sys), sys.get_basis());
 gen_op FY = gen_op(Fy(sys), sys.get_basis());
 gen_op FZ = gen_op(Fz(sys), sys.get_basis());

 gen_op sigmaeq = sigma_eq(sys);        // Density matrix for equilibrium
 Mx = Re(trace(FX, sigmaeq));           // Get x-magnetization component
 My = Re(trace(FY, sigmaeq));           // Get y-magnetization component
 Mz = Re(trace(FZ, sigmaeq));           // Get z-magnetization component
 cout << “\n\nEquilibruim Magnetization: “
      << Mx << “\t” << My << “\t” << Mz;
 gen_op sigma=Ixpuls(sys,sigmaeq,-45);  // Apply an (PI/2)x pulse
 Mx = Re(trace(FX, sigma));             // Get x-magnetization component
 My = Re(trace(FY, sigma));             // Get y-magnetization component
 Mz = Re(trace(FZ, sigma));             // Get z-magnetization component
 cout << “\n\nMagnetization After Pulse: “
      << Mx << “\t” << My << “\t” << Mz;

 int npts = 501;                        // Number of magnetization points
 block_1D Mdata(npts);                  // For storing M x & y components
 double ttot = 1.e-8;                   // Total time for 5 cycles
 double tinc = ttot/double(npts-1);     // Time increment between points

//      Begin Looping Over Different Times

 double time;
 gen_op sigmat;
 for(int j=0; j<npts; j++)
   {
   time = double(j)*tinc;               // Total applied field length
   sigmat = evolve(sigma, H, time);     // Evolve the Zeeman Hamiltonian
   Mx = Re(trace(FX, sigmat));          // Get x-magnetization component
   My = Re(trace(FY, sigmat));          // Get y-magnetization component
   Mz = Re(trace(FZ, sigmat));          // Get z-magnetization component
   Mdata(j) = complex(Mx,My);           // Store x & y magnetization
   cout << “\n”<< Mx << “\t”
        << My << “\t” << Mz;
   }
 FM_1D(“Zeeman.mif”,Mdata,14,5,0,1,2);  // Output mag. plot, FM format
 cout << “\n”;                          // End with screen nice
 }
```

# Heff.cc

## *Effective Hamiltonian*

```
/* Heff.cc **************************************-*-c++-*-
**                                                      **
**        Example program for the GAMMA Library         **
**                                                      **
** Monitors Heff eigenvals vs. B1 in a 2 spin 1H system **
**                                                      **
*********************************************************/

#include “gamma.h”

main (int argc, char* argv[])
{
 String filename;                       // Name of spin system file
 query_parameter(argc, argv, 1          // Get filename from command
 “\nSpin system filename? “, filename); // line or ask for them
 sys_dynamic sys;                       // Declare dynamic spin system
 sys.read(filename);                    // Read system from filename
 int size = 1001;                       // Plot 1001 pts each eigenvalue
 double gB1inc = 10;                    // Increment gamma*B1 by 10 Hertz
 double Wrf = 0;                        // Keep rf-field at 0 Hertz
 double gamB1 = 0;                      // Initial rf-field strength (Hz)
 gen_op Ho = Hcs(sys) + HJ(sys);        // Set the Hamiltonian
 int hs = Ho.dim();                     // Get Hilbert space dimension
 row_vector plots[hs], plot(size);      // Storage for eigenvalue plots
 double w[hs];                          // Storage for eigenvalues
 for(int k=0; k<hs; k++)                // Set up for individual plots
   plots[k] = plot;
 gen_op H;                              // Effective Hamiltonian
 for(int i=0; i<size; i++)
   {
   H = Heff(sys, Ho, “1H”, Wrf, gamB1); // Effective Ham (Wrf = 0 Hz)
   H.eigvals(w, 1);                     // Get eigenvalues, ordered
   gamB1 += gB1inc;                     // Increment gamma*B1
   for(int k=0; k<hs; k++)              // Store eigenvalues for this B1
     (plots[k]).put(w[k], i);
   }
 FM_1Dm(“eigval.mif”, hs, plots, 19.0,  // Plot all eigenvalues vs. B1
          14.0, 0.0, (size-1)*gB1inc);
 cout << “\n”;
 }
```

## Dip_SpinT.cc

### *Dipolar Hamiltonian Spin Tensors*

```
/* Dip_SpinT.cc *****************************************-*-c++-*-
**                                                              **
**      Example program for the GAMMA Library                   **
**                                                              **
** This program computes the dipolar Hamiltionian rank 2 spin   **
** tensor. 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.                                           **
** Note: Although usually only the rank 2 components are used   **
** (and returned by the function T_D) this routine also         **
** deals with the rank 0 and rank 1 components as well.         **
**                                                              **
******************************************************************/

#include <gamma.h>

void DCompOut(matrix& SOp, int l, int m, complex& fact)

 {
 cout << “\n”
 << “\n\t\t D”
 << “\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 Dipolar Hamiltonain Spin Tensors\n\n”;
 sys_dynamic dsys(2);                   // Set a 2 spin system
 spin_T TD = T_D(dsys, 0, 1);           // Get dipolar spin tensor
 spin_T TDF = 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(TDF.component(0,0)); // Get 00 component
 DCompOut(SOp, 0, 0, fact);             // Output to screen
 FM_Matrix(“Dip00.mmf”, SOp);           // Output T00 to FrameMaker
 fact = -2.0*sqrt(2.0);                 // Set scaling factor
 SOp = fact*matrix(TDF.component(1,0)); // Get 10 component
 DCompOut(SOp, 1, 0, fact);             // Output to screen
 FM_Matrix(“Dip10.mmf”, SOp);           // Output T10 to FrameMaker
 fact = -4.0;                           // Set scaling factor
 SOp = fact*matrix(TDF.component(1,1)); // Get 11 component
 DCompOut(SOp, 1, 1, fact);             // Output to screen
 FM_Matrix(“Dip11.mmf”, SOp);           // Output T11 to FrameMaker
 SOp = fact*matrix(TDF.component(1,-1));// Get 1-1 component
 DCompOut(SOp, 1, -1, fact);            // Output to screen
 FM_Matrix(“Dip1m1.mmf”, SOp);          // Output T1-1 to FrameMaker
 fact = 2.0*sqrt(6.0);                  // Set scaling factor
 SOp = fact*matrix(TD.component(2,0));  // Get 20 component
 DCompOut(SOp, 2, 0, fact);             // Output to screen
 FM_Matrix(“Dip20.mmf”, SOp);           // Output T20 to FrameMaker
 fact = 4.0;                            // Set scaling factor
 SOp = fact*matrix(TD.component(2,1));  // Get 21 component
 DCompOut(SOp, 2, 1, fact);             // Output to screen
 FM_Matrix(“Dip21.mmf”, SOp);           // Output T21 to FrameMaker
 SOp = fact*matrix(TD.component(2,-1)); // Get 2-1 component
 DCompOut(SOp, 2, -1, fact);            // Output to screen
 FM_Matrix(“Dip2m1.mmf”, SOp);          // Output T2-1 to FrameMaker
 fact = 2.0;                            // Set scaling factor
 SOp = fact*matrix(TD.component(2,2));  // Get 22 component
 DCompOut(SOp, 2, 2, fact);             // Output to screen
 FM_Matrix(“Dip22.mmf”, SOp);           // Output T22 to FrameMaker
 SOp = fact*matrix(TD.component(2,-2)); // Get 2-2 component
 DCompOut(SOp, 2, -2, fact);            // Output to screen
 FM_Matrix(“Dip2m2.mmf”, SOp);          // Output T2-2 to FrameMaker
 cout << “\n\n”;                        // Keep screen nice
 }
```

# Quad_SpinT.cc

## *Quadrupolar Hamiltonian Spin Tensors*

```
/* Quad_SpinT.cc *****************************************-*-c++-*-
**                                                              **
** “Quadrupolar” Irreducible Spin Tensor Components             **
**                                                              **
** This program computes the quadrupolar Hamiltionian (rank 2)  **
** irreducible spin tensor compnents. The 9 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.                 **
**                                                              **
** The components shown are for a single spin with I=1 in a     **
** product basis.                                               **
**                                                              **
** Note: Although usually only the rank 2 components are used   **
** (and returned by the function T_Q) this routine deals        **
** with the rank 0 and rank 1 components as well.               **
**                                                              **
** Note: These spin tensor components are “quadrupolar” only    **
** because they are rank 2 & involve two contributions          **
** from the same spin. The “dipolar” components involve         **
** two different spins and the rank 2 “shielding anisot.”       **
** has one component from spin and one from a static            **
** magnetic field vector. All stem from the same source.        **
**                                                              **
** Author: S.A. Smith                                           **
** Last Update: Jan. 11 1993                                    **
**                                                              **
*****************************************************************/

#include <spin_T.h>
#include <FrameMaker.h>

void QCompOut(matrix& SOp, int l, int m, complex& fact)

 {
 cout << “\n”
 << “\n\t\t Q”
 << “\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 Quadrupolar Hamiltonain Spin Tensors\n\n”;
 spin_sys sys(1);                      // Set a single spin system
 sys.isotope(0,”2H”);                  // Set spin to deuterium
 spin_T TQ = T2(sys, 0, 0);            // Get full rank 2 spin tensor
 matrix SOp;                           // Working spin operator
 complex fact = -0.5*sqrt(3.0);        // Set scaling factor
 SOp = fact*matrix(TQ.component(0,0));  // Get 00 component
 QCompOut(SOp, 0, 0, fact);            // Output to screen
 FM_Matrix(“Quad00.mmf”, SOp);         // Output T00 to FrameMaker
 fact = -1.0*sqrt(2.0);                // Set scaling factor
 SOp = fact*matrix(TQ.component(1,0));  // Get 10 component
 QCompOut(SOp, 1, 0, fact);            // Output to screen
 FM_Matrix(“Quad10.mmf”, SOp);         // Output T10 to FrameMaker
 fact = -sqrt(2.0);                    // Set scaling factor
 SOp = fact*matrix(TQ.component(1,1));  // Get 11 component
 QCompOut(SOp, 1, 1, fact);            // Output to screen
 FM_Matrix(“Quad11.mmf”, SOp);         // Output T11 to FrameMaker
 SOp = fact*matrix(TQ.component(1,-1)); // Get 1-1 component
 QCompOut(SOp, 1, -1, fact);           // Output to screen
 FM_Matrix(“Quad1m1.mmf”, SOp);        // Output T1-1 to FrameMaker
 fact = sqrt(6.0);                     // Set scaling factor
 SOp = fact*matrix(TQ.component(2,0));  // Get 20 component
 QCompOut(SOp, 2, 0, fact);            // Output to screen
 FM_Matrix(“Quad20.mmf”, SOp);         // Output T20 to FrameMaker
 fact = sqrt(2.0);                     // Set scaling factor
 SOp = fact*matrix(TQ.component(2,1));  // Get 21 component
 QCompOut(SOp, 2, 1, fact);            // Output to screen
 FM_Matrix(“Quad21.mmf”, SOp);         // Output T21 to FrameMaker
 SOp = fact*matrix(TQ.component(2,-1)); // Get 2-1 component
 QCompOut(SOp, 2, -1, fact);           // Output to screen
 FM_Matrix(“Quad2m1.mmf”, SOp);        // Output T2-1 to FrameMaker
 fact = 1.0;                           // Set scaling factor
 SOp = fact*matrix(TQ.component(2,2));  // Get 22 component
 QCompOut(SOp, 2, 2, fact);            // Output to screen
 FM_Matrix(“Quad22.mmf”, SOp);         // Output T22 to FrameMaker
 SOp = fact*matrix(TQ.component(2,-2)); // Get 2-2 component
 QCompOut(SOp, 2, -2, fact);           // Output to screen
 FM_Matrix(“Quad2m2.mmf”, SOp);        // Output T2-2 to FrameMaker
 cout << “\n\n”;                       // Keep screen nice
 }
```

# 9 Ideal Pulses

## 9.1 Overview

Ideal pulses evolve a spin system under the effects of an ***ideal*** hard pulse. This is performed in the limit that the applied pulse is infinitely strong and infinitely short. All spins affected by an ideal pulse are rotated with by the same angle and phase. Since they are instantaneous, they cannot include the effects of relaxation and/or exchange. Any pulse angle, phase, or selectivity is allowed.

These pulses are often the first type used when building a simulation program. The pulses are easy to use and will not in themselves produce any odd phasing effects from pulse length or pulse strength. Once a program using ideal pulses has been successfully produced, the pulses may be replaced with other pulse types that more closely mimic experiments[1].

## 9.2 Available Functions

**Ideal Pulses**

| | | |
|---|---|---|
| Ixpuls | - Pulses on the x-axis | page 272 |
| Iypuls | - Pulses on the y-axis | page 274 |
| Ixypuls | - Pulses in the xy-plane | page 276 |

**Pulse Propagators**

| | | |
|---|---|---|
| Ixpuls_U | - Propagator for pulse on the x-axis | page 278 |
| Iypuls_U | - Propagator for pulse on the y-axis | page 280 |
| Ixypuls_U | - Propagator for pulse in the xy-plane | page 282 |

## 9.3 Figures & Tables

| | |
|---|---|
| X-Axis Pulse Rotation Direction | page 272 |
| Y-Axis Pulse Rotation Direction | page 274 |
| Phased Pulse Rotation Direction | page 276 |

[1] Note that ideal pulses can be set to perform pulses that are inconsistent with quantum mechanics. In particular, one may apply a selective pulse on a single spin even when that spin is indistinguishable from another spin! You can pulse one proton of a rapidly rotating methyl group for example even though that is nonsensical.

# 9.4 Pulse Routines

## 9.4.1 Ixpuls

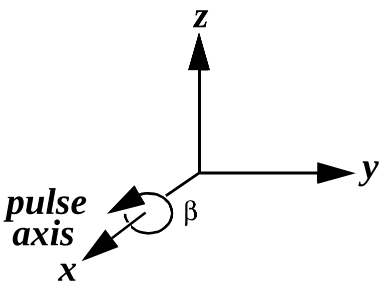
### Usage:

```
#include <gamma.h >
gen_op Ixpuls (spin_system &sys, gen_op& sigma, double beta)
gen_op Ixpuls(spin_system &sys, gen_op& sigma, int spin, double beta)
gen_op Ixpuls (spin_system &sys, gen_op& sigma, char *iso, double beta)
gen_op Ixpuls_sp (spin_system &sys, gen_op& sigma, double beta)
```

### Description:

Applies an ideal pulse to the input density operator ***sigma*** associated with the spin system ***sys***. The pulse is applied along the x-axis and the rotation angle is ***beta*** degrees. Any other arguments set the pulse selectivity. Spin rotation occurs about the x-axis as depicted in the following diagram.

***X-Axis Pulse Rotation Direction***



1. Ixpuls(spin_system &sys, gen_op &sigma, double beta) - Pulse is applied to all spins in the spin system.
2. Ixpuls(spin_system &sys, gen_op &sigma, int spin, double beta) - Pulse is applied only to one spin ***spin***.
3. Ixpuls(spin_system &sys, gen_op &sigma, string iso, double beta) - Pulse is applied to all spins of the isotope type specified by ***iso*** (e.g. “1H”, “13C”, “19F”).
4. Ixpuls_sp(spin_system &sys, gen_op &sigma, double beta) - Pulse is applied only to spins which have had their spin flags set to TRUE prior to the function call (see Class SpinSys).

The pulse angle “beta” is specified in degrees and has a default value of 90 if left out of the argument list.

### Return Value:

The function returns a general operator.

### Examples:

```
#include <gamma.h >
main()
  {
  gen_op sigma;                          // set up a general operator for the density matrix.
  spin_system AMX(3);                    // set up a three spin system.
  AMX.isotope(0, “1H”);                  // set the first spin to a proton.
  AMX.isotope(1, “19F”);                 // set the second spin to a fluorine.
  AMX.isotope(2, “19F”);                 // set the third spin to a fluorine.
  sigma = sigma_eq(AMX);                 // set the density matrix to equilibrium for AMX
  sigma = Ixpuls(AMX, sigma, 90.0)      ;    // Pulse all spins by 90 degrees.
  sigma = Ixpuls(AMX, sigma, 1, 180.0);    // Pulse the second spin by 180 degrees.
  sigma = Ixpuls(AMX, sigma, “19F”, 90.0);// Pulse all fluorines by 90 degrees.
  AMX.sp_flags_FALSE();                     // insure all spins have flags set to FALSE.
  AMX.sp_flag(“1H”, TRUE);                  // set all proton flags to TRUE (only spin 0).
  AMX.sp_flag(1, TRUE);                     // set spin 1 flag to TRUE (first fluorine).
  sigma = Ixpuls_sp(AMX, sigma, 45.0);     // Pulse the proton and the first fluorine by 45 degrees.
  }
```

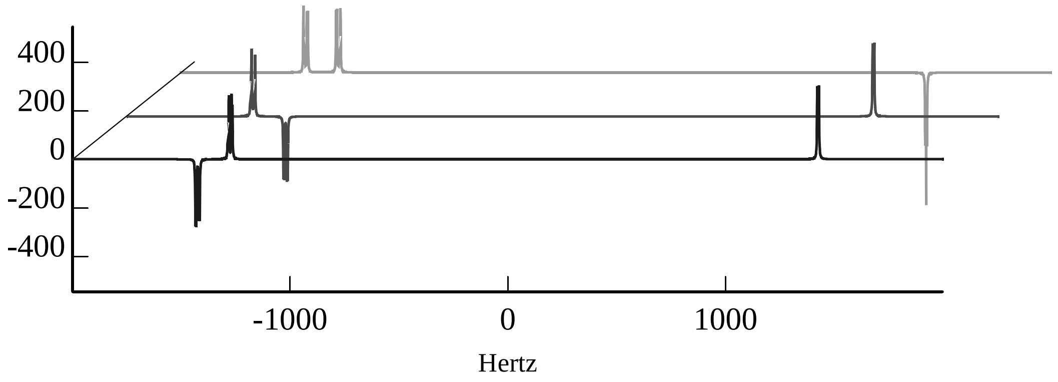
**Mathematical Basis:**

$$\sigma^{IP} = \begin{cases} \boldsymbol{R}_x(\beta)\sigma_0[\boldsymbol{R}_x(\beta)]^{-1} & \\ \boldsymbol{R}_{i,x}(\beta)\sigma_0[\boldsymbol{R}_{i,x}(\beta)]^{-1} & \\ \boldsymbol{R}_{\{i\},x}(\beta)\sigma_0[\boldsymbol{R}_{i,x}(\beta)]^{-1} & j \in \{i\} \forall\ j \ni \gamma_j = \gamma_i \\ \boldsymbol{R}_{\{i\},x}(\beta)\sigma_0[\boldsymbol{R}_{i,x}(\beta)]^{-1} & j \in \{i\} \forall\ j \ni flag_j = TRUE_i \end{cases}$$

**See Also: Ixpuls_U, Iypuls, Ixypuls, Sxpuls**

The following plot is of a three spin system following a spin selective $\pi_x$ ideal pulse followed by a $(\pi/2)_y$ ideal pulse. The GAMMA code which produced this is given at the end of the chapter. The sequence is successively performed with the selective pulse choosing a different spin in the system.

**Effect of Spin Selective Ideal x Pulse (2)**



## 9.4.2 Iypuls

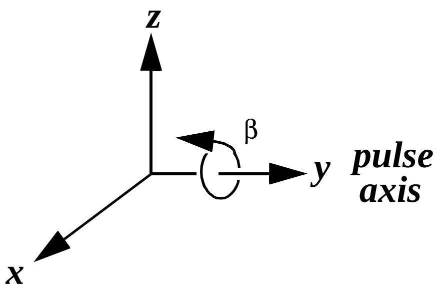
### Usage:

```
#include <gamma.h >
gen_op Iypuls(const spin_system &sys, gen_op& sigma, double beta)
gen_op Iypul(const spin_system &sys, gen_op& sigma, int spin, double beta)
gen_op Iypuls(const spin_system &sys, gen_op& S, string iso, double beta)
gen_op Iypuls_sp(const spin_system &sys, gen_op& sigma, double beta)
```

### Description:

Applies an ideal pulse to the input density operator ***sigma*** associated with the spin system ***sys***. The pulse is applied along the y-axis and the rotation angle is ***beta*** degrees. Any other arguments set the pulse selectivity. The input density operator remains unchanged and the function returns a new density operator representing the state of the system following the pulse applied on ***sigma***. Spin rotation occurs about the y-axis as depicted in the following diagram.

***Y-Axis Pulse Rotation Direction***



1. Iypuls(spin_system &sys, gen_op &sigma, double beta) - Pulse is applied to all spins in the spin system.
2. Iypuls(spin_system &sys, gen_op &sigma, int spin, double beta) - Pulse is applied only to one spin ***spin***.
3. Iypuls(spin_system &sys, gen_op &sigma, string iso, double beta) - Pulse is applied to all spins of the isotope type specified by ***iso*** (e.g. “1H”, “13C”, “19F”).
4. Iypuls_sp(spin_system &sys, gen_op &sigma, double beta) - Pulse is applied only to spins which have had their spin flags set to TRUE prior to the function call (see Class SpinSys).

The pulse angle “beta” is specified in degrees and has a default value of 90 if left out of the argument list.

### Return Value:

The function returns a general operator.

### Examples:

```
include <gamma.h>
main()
  {
  gen_op sigma;                        // set up a general operator for the density matrix.
  spin_system AMX(3);                  // set up a three spin system.
  AMX.isotope(0, “1H”);                // set the first spin to a proton.
  AMX.isotope(1, “19F”);               // set the second spin to a fluorine.
  AMX.isotope(2, “19F”);               // set the third spin to a fluorine.
  sigma = sigma_eq(AMX);               // set the density matrix to equilibrium for AMX
  sigma = Iypuls(AMX, sigma, 90.0);    // Pulse all spins by 90 degrees.
  sigma = Iypuls(AMX, sigma, 1, 180.0);    // Pulse the second spin by 180 degrees.
  sigma = Iypuls(AMX, sigma, “19F”, 90.0);// Pulse all fluorines by 90 degrees.
  AMX.sp_flags_FALSE();                    // insure all spins have flags set to FALSE.
  AMX.sp_flag(“1H”, TRUE);                 // set all proton flags to TRUE.
  AMX.sp_flag(1, TRUE);                    // set spin 1 flag to TRUE.
  sigma = Iypuls_sp(AMX, sigma, 45.0);     // Pulse the proton and the first fluorine by 45 degrees.
  }
```

**Mathematical Basis:**

$$\sigma^{IP} = \begin{array}{ll} \boldsymbol{R}_y(\beta)\sigma_0[\boldsymbol{R}_y(\beta)]^{-1} & \\ \boldsymbol{R}_{i,y}(\beta)\sigma_0[\boldsymbol{R}_{i,y}(\beta)]^{-1} & \\ \boldsymbol{R}_{\{i\},y}(\beta)\sigma_0[\boldsymbol{R}_{i,y}(\beta)]^{-1} & j \in \{i\}\forall\ j \ni \gamma_j = \gamma_i \\ \boldsymbol{R}_{\{i\},y}(\beta)\sigma_0[\boldsymbol{R}_{i,y}(\beta)]^{-1} & j \in \{i\}\forall\ j \ni flag_j = TRUE_i \end{array}$$

**See Also: Iypuls_U, Ixpuls, Ixypuls, Sypul**

## 9.4.3 Ixypuls

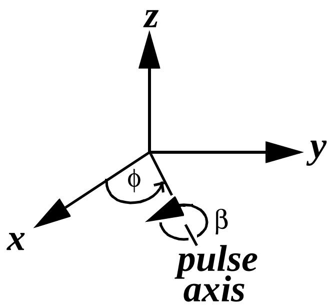
**Usage:**

```
#include <gamma.h >
gen_op Ixypuls (spin_system &sys, gen_op& sigma, double beta, double phi)
gen_op Ixypuls(spin_system &sys, gen_op& sigma, int spin, double beta,
double phi)
gen_op Ixypuls (spin_system &sys, gen_op& sigma, char *iso, double beta,
double phi)
gen_op Ixypuls_sp (spin_system &sys, gen_op& sigma, double beta, double
phi)
```

**Description:**

Applies an “ideal pulse” to the density matrix of angle beta about an axis in the xy-plane phi degrees over from the x-axis. The pulse affects only the spins specified.

***Phased Pulse Rotation Direction***



1. Ixypuls(spin_system &sys, gen_op &sigma, double beta, double phi) - Returns the density matrix operator, sigma, after having rotation of angle beta about an axis ϕ degrees from the x-axis applied to all spins in the spin system.
2. Ixypuls(spin_system &sys, gen_op &sigma, int spin, double beta, double phi) -Returns the density matrix operator, sigma, after having rotation of angle beta about an axis ϕ degrees from the x-axis applied to only the spin specified.
3. Ixypuls(spin_system &sys, gen_op &sigma, char *iso, double beta, double phi) - Returns the density matrix operator, sigma, after having rotation of angle beta about an axis ϕ degrees from the x-axis applied to the spins of the isotope type specified.
4. Ixypuls_sp(spin_system &sys, gen_op &sigma, double phi, double beta) - Returns the density matrix operator, sigma, after having been “pulsed” with a rotation of angle beta about an axis ϕ degrees from the x-axis. The pulse is applied only to spins which have had their spin flags set to TRUE to the function call (see Class Spin System).

The pulse phase angle “phi” is specified in degrees and has a default value of 0 if left out of the argument list.

The pulse angle “beta” is specified in degrees and has a default value of 90 if both it and phi are left out of the argument list.

**Return Value:**

The function returns a general operator.

**Example(s):**

```
#include <gamma.h >
main()
  {
  gen_op sigma;                          // set up a general operator for the density matrix.
  spin_system AMX(3);                    // set up a three spin system.
  AMX.isotope(0, “1H”);                  // set the first spin to a proton.
  AMX.isotope(1, “19F”);                 // set the second spin to a fluorine.
  AMX.isotope(2, “19F”);                 // set the third spin to a fluorine.
  sigma = sigma_eq(AMX);                 // set the density matrix to equilibrium for AMX
  sigma = sigma_eq(AMX);                  // set the density matrix to equilibrium for AMX
  sigma = Ixypuls(AMX, sigma, 90.0, 270.0);// Pulse all spins by 90 degrees on the -y axis.
  sigma = Ixypuls(AMX, sigma, 1, 180.0, 180.0);// Pulse the second spin by 180 degrees along -x.
  sigma = Ixypuls(AMX, sigma, “19F”, 90.0, 90.);// Pulse all fluorines by 90 degrees on the y axis.
  AMX.sp_flags_FALSE();                   // insure all spins have flags set to FALSE.
  AMX.sp_flag(“1H”, TRUE);                // set all proton flags to TRUE.
  AMX.sp_flag(1, TRUE);                  // set spin 1 flag to TRUE.
  sigma = Ixypuls_sp(AMX, sigma, 45.0, 0.0);// Pulse proton & the first fluorine by 45 degrees on x.
  }
```

**Mathematical Basis:**

$$\sigma^{IP} = \begin{array}{ll} \boldsymbol{R}_{xy}(\phi, \beta)\sigma_0[\boldsymbol{R}_{xy}(\phi, \beta)]^{-1} & \\ \boldsymbol{R}_{i,\,xy}(\phi, \beta)\sigma_0[\boldsymbol{R}_{i,\,xy}(\phi, \beta)]^{-1} & \\ \boldsymbol{R}_{\{i\},\,xy}(\phi, \beta)\sigma_0[\boldsymbol{R}_{\{i\},\,xy}(\phi, \beta)]^{-1} & j \in \{i\} \forall\ j \ni \gamma_j = \gamma_i \\ \boldsymbol{R}_{\{i\},\,xy}(\phi, \beta)\sigma_0[\boldsymbol{R}_{\{i\},\,xy}(\phi, \beta)]^{-1} & j \in \{i\} \forall\ j \ni flag_j = TRUE_i \end{array}$$

**See Also: Ixypuls_U, Ixpuls, Iypuls, Sxypul**

# 9.5 Propagator Routines

## 9.5.1 Ixpuls_U

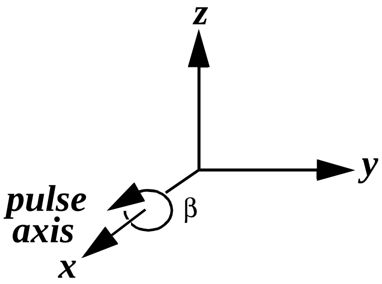
**Usage:**

```
#include <HSLib/PulseI.h >
gen_op Ixpuls_U(const spin_system &sys, double beta)
gen_op Ixpuls_U(const spin_system &sys, int spin, double beta)
gen_op Ixpuls_U(const spin_system &sys, char *iso, double beta)
gen_op Ixpuls_U _sp(const spin_system &sys, double beta)
```

**Description:**

Constructs a propagator for an ideal pulse associated with the spin system ***sys***. The pulse is applied along the x-axis and the rotation angle is ***beta*** degrees. Any other arguments set the pulse selectivity. Spin rotation occurs about the x-axis as depicted in the following diagram.

***X-Axis Pulse Rotation Direction***



1. Ixpuls_U (spin_system &sys, double beta) - Returns the propagator for an ideal pulse with a rotation angle beta about the x-axis. The propagator affects all spins in the spin system.
2. Ixpuls_U (spin_system &sys, int spin, double beta) - Returns the propagator for an ideal pulse with a rotation angle beta about the x-axis. The propagator affects only to the spin specified.
3. Ixpuls_U (spin_system &sys, char *iso, double beta) - Returns the propagator for an ideal pulse with a rotation angle beta about the x-axis. The propagator affects only to spins of the isotope type specified.
4. Ixpuls_U _sp(spin_system &sys, double beta) - Returns the propagator for an ideal pulse with a rotation angle beta about the x-axis. The propagator affects only to spins which have had their spin flags set to TRUE prior to the function call (see Class Spin System).

The pulse angle “beta” is specified in degrees and has a default value of 90 if left out of the argument list.

**Return Value:**

The function returns a general operator.

**Examples:**

```
#include <gamma.h >
main()
  {
  gen_op U;                          // set up a general operator for the ideal pulse propagator.
  spin_system AMX(3);                // set up a three spin system.
  AMX.isotope(0, “1H”);                 // set the first spin to a proton.
  AMX.isotope(1, “19F”);                // set the second spin to a fluorine.
  AMX.isotope(2, “19F”);                // set the third spin to a fluorine.
  U = Ixpuls_U (AMX, 90.0);             // U set to propagator for pulse on all spins by 90 degrees.
  U = Ixpuls_U (AMX, 1, 180.0);         // U set to propagator for 180 pulse on the second spin de-
  grees.
  U = Ixpuls_U (AMX, “19F”, 90.0);      // U set to propagator for pulse on all fluorines by 90 degrees.
  AMX.sp_flags_FALSE();                 // insure all spins have flags set to FALSE.
  AMX.sp_flag(“1H”, TRUE);              // set all proton flags to TRUE (only spin 0).
  AMX.sp_flag(1, TRUE);                 // set spin 1 flag to TRUE (first fluorine).
  U = Ixpuls_U _sp(AMX, 45.0);          // U pulse propagator on proton & first fluorine by 45 degrees.
  }
```

**Mathematical Basis:**

$$\boldsymbol{U}^{IP} = \begin{array}{ll} \boldsymbol{R}_x(\beta) & \\ \boldsymbol{R}_{i,\,x}(\beta) & \\ \boldsymbol{R}_{\{i\},\,x}(\beta) & \qquad j \in \{i\} \forall \;\; j \ni \gamma_j = \gamma_i \\ \boldsymbol{R}_{\{i\},\,x}(\beta) & \qquad j \in \{i\} \forall \;\; j \ni flag_j = TRUE_i \end{array}$$

**See Also: Ixpuls, Iypuls_U, Ixypuls_U, Sxpuls**

## 9.5.2 Iypuls_U

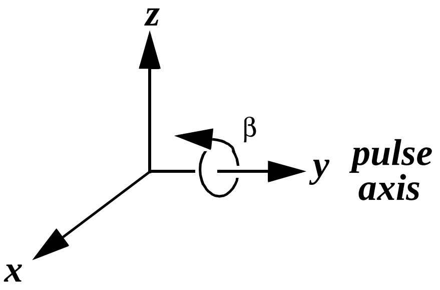
### Usage:

```
#include <gamma.h >
gen_op Iypuls_U  (spin_system &sys, double beta)
gen_op Iypuls_U (spin_system &sys, int spin, double beta)
gen_op Iypuls_U (spin_system &sys, char *iso, double beta)
gen_op Iypuls_U _sp (spin_system &sys, double beta)
```

### Description:

Constructs a propagator for an ideal pulse associated with the spin system ***sys***. The pulse is applied along the y-axis and the rotation angle is ***beta*** degrees. Any other arguments set the pulse selectivity. Spin rotation occurs about the x-axis as depicted in the following diagram.

***Y-Axis Pulse Rotation Direction***



1. Iypuls_U (spin_system &sys, double beta) - Returns the propagator for an ideal pulse with a rotation angle beta about the y-axis. The propagator affects all spins in the spin system.
2. Iypuls_U (spin_system &sys, int spin, double beta) - Returns the propagator for an ideal pulse with a rotation angle beta about the y-axis. The propagator affects only to the spin specified.
3. Iypuls_U (spin_system &sys, char *iso, double beta) - Returns the propagator for an ideal pulse with a rotation angle beta about the y-axis. The propagator affects only to spins of the isotope type specified.
4. Iypuls_U _sp(spin_system &sys, double beta) - Returns the propagator for an ideal pulse with a rotation angle beta about the y-axis. The propagator affects only to spins which have had their spin flags set to TRUE prior to the function call (see Class Spin System).

The pulse angle “beta” is specified in degrees and has a default value of 90 if left out of the argument list.

### Return Value:

The function returns a general operator.

### Examples:

```
#include <gamma.h >
main()
  {
  gen_op U;                        // set up a general operator for the ideal pulse propagator.
  spin_system AMX(3);              // set up a three spin system.
  AMX.isotope(0, “1H”);            // set the first spin to a proton.
  AMX.isotope(1, “19F”);           // set the second spin to a fluorine.
  AMX.isotope(2, “19F”);           // set the third spin to a fluorine.
  U = Iypuls_U (AMX, 90.0);        // U set to propagator for pulse on all spins by 90 degrees.
  U = Iypuls_U (AMX, 1, 180.0);            // U set to propagator for 180 pulse on the second spin de-
  grees.
  U = Iypuls_U (AMX, “19F”, 90.0);         // U set to propagator for pulse on all fluorines by 90 degrees.
  AMX.sp_flags_FALSE();                    // insure all spins have flags set to FALSE.
  AMX.sp_flag(“1H”, TRUE);                 // set all proton flags to TRUE (only spin 0).
  AMX.sp_flag(1, TRUE);                    // set spin 1 flag to TRUE (first fluorine).
  U = Iypuls_U _sp(AMX, 45.0);             // U pulse propagator on proton & first fluorine by 45 degrees.
  }
```

**Mathematical Basis:**

$$\boldsymbol{U}^{IP} = \begin{array}{ll} \boldsymbol{R}_{y}(\beta) & \\ \boldsymbol{R}_{i,\,y}(\beta) & \\ \boldsymbol{R}_{\{i\},\,y}(\beta) & j \in \{i\} \forall\ j \ni \gamma_j = \gamma_i \\ \boldsymbol{R}_{\{i\},\,y}(\beta) & j \in \{i\} \forall\ j \ni flag_j = TRUE_i \end{array}$$

**See Also: Iypuls, Ixpuls_U, Ixypuls_U, Sypuls**

### 9.5.3 Ixypuls_U

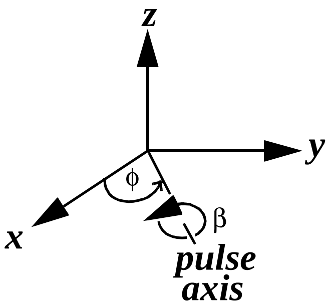
**Usage:**

```
#include <gamma.h >
gen_op Ixypuls_U  (spin_system &sys, double beta, double phi)
gen_op Ixypuls_U (spin_system &sys, int spin, double beta, double phi)
gen_op Ixypuls_U  (spin_system &sys, char *iso, double beta, double phi)
gen_op Ixypuls_U _sp (spin_system &sys, double beta, double phi)
```

**Description:**

Constructs a propagator for an ideal pulse associated with the spin system ***sys***. The pulse is applied along the an axis in the xy-plane that is ***phi*** degrees from the x-axis and the rotation angle is ***beta*** degrees. Any other arguments set the pulse selectivity. Spin rotation occurs about the x-axis as depicted in the following diagram.

***Phased Pulse Rotation Direction***



1. Ixypuls_U(spin_system &sys, double beta, double phi) - Ideal pulse propagator active on all spins.
2. Ixypuls_U(spin_system &sys, int spin, double beta, double phi) - Ideal pulse propagator active only on the spin specified.
3. Ixypuls_U (spin_system &sys, char *iso, double beta, double phi) - Returns the propagator for an ideal pulse with rotation of angle beta about an axis ϕ degrees from the x-axis applied to the spins of the isotope type specified.
4. Ixypuls_U _sp(spin_system &sys, double beta, double phi) - Returns the propagator for an ideal pulse with a rotation of angle beta about an axis ϕ degrees from the x-axis. The pulse is applied only to spins which have had their spin flags set to TRUE to the function call (see Class Spin System).

The pulse phase angle “phi” is specified in degrees and has a default value of 0 if left out of the argument list.

The pulse angle “beta” is specified in degrees and has a default value of 90 if both it and phi are left out of the argument list.

**Return Value:**

The function returns a general operator.

**Examples:**

```
#include <gamma.h >
main()
  {
  gen_op U;                                  // set up a general operator for the ideal pulse propagator.
  spin_system AMX(3);                        // set up a three spin system.
  AMX.isotope(0, “1H”);                      // set the first spin to a proton.
  AMX.isotope(1, “19F”);                   // set the second spin to a fluorine.
  AMX.isotope(2, “19F”);                   // set the third spin to a fluorine.
  U = Ixypuls_U (AMX, 90.0, 270.0);        // U propagator for 90 pulse on all spins on the -y axis.
  U = Ixypuls_U (AMX, 1, 180.0, 180.0);    // U propagator for 180 pulse on the second spin along -x.
  U = Ixypuls_U (AMX, “19F”, 90.0, 90.0);  // U propagator for 90 pulse on all fluorines on the y axis.
  AMX.sp_flags_FALSE();                    // insure all spins have flags set to FALSE.
  AMX.sp_flag(“1H”, TRUE);                 // set all proton flags to TRUE (only spin 0).
  AMX.sp_flag(1, TRUE);                    // set spin 1 flag to TRUE (first fluorine).
  U = Ixypuls_U _sp(AMX, 45.0, 0.0);       // U x-pulse propagator on proton & first fluorine by 45 degrees.
  }
```

**Mathematical Basis:**

$$\boldsymbol{U}^{IP} = \begin{array}{ll} \boldsymbol{R}_{xy}(\beta) & \\ \boldsymbol{R}_{i,\,xy}(\beta) & \\ \boldsymbol{R}_{\{i\},\,xy}(\beta) & j \in \{i\} \forall\ j \ni \gamma_j = \gamma_i \\ \boldsymbol{R}_{\{i\},\,xy}(\beta) & j \in \{i\} \forall\ j \ni flag_j = TRUE_i \end{array}$$

**See Also: Ixypuls, Ixpuls_U, Iypuls_U, Sxypul**

## 9.6 Description

The effective Hamiltonian for a spin system under the effects of an applied rf-field is given by

$$\boldsymbol{H}_{eff} = \boldsymbol{H}_o - \Omega_{rf}\boldsymbol{F}_z + B_{rf}\boldsymbol{F}_\phi \qquad (9\text{-}1)$$

where operator $\boldsymbol{F}_\phi$ is the spin operator $\boldsymbol{F_x}$ rotated about the z-axis by phase angle $\phi$.

$$\boldsymbol{F}_\phi = \boldsymbol{R}_z(\phi)\boldsymbol{F}_x\boldsymbol{R}_z(\phi)$$

For a single spin with the rf-field on resonance, two of the terms in the effective Hamiltonian exactly cancel, $\boldsymbol{H}_o - \Omega_{rf}\boldsymbol{F}_z = 0$ leaving only $\boldsymbol{H}_{eff} = B_{rf}\boldsymbol{F}_\phi$ and effective propagator identical to that of an ideal pulse

As this last transformation is simply a rotation about the z-axis by an angle $\theta_{rf}$, the final result for propagation of the density matrix through a real “soft” pulse is

$$\sigma^{SP} = \boldsymbol{R}_z(\theta_{rf})[e^{2\pi i H_{eff}t}]\sigma_0([e^{-2\pi i H_{eff}t}][\boldsymbol{R}_z(\theta_{rf})])^{-1} \quad , \qquad (9\text{-}2)$$

or equivalently

$$\sigma^{SP} = \boldsymbol{U}^{SP}\sigma_0[\boldsymbol{U}^{SP}]^{-1} \text{ with the soft pulse propagator given by } \boldsymbol{U}^{SP} = \boldsymbol{R}_z(\theta_{rf})e^{-2\pi i H_{eff}t_p} \qquad (9\text{-}3)$$

where

$$\theta_{rf} = \gamma B_{rf}t_p \qquad \text{and} \qquad \boldsymbol{H}_{eff} = \boldsymbol{H}_o + \Omega_{rf}\boldsymbol{F}_z - B_{rf}\boldsymbol{F}_{xy}(\varphi) \qquad (9\text{-}4)$$

### *Hard “Ideal” Pulses*

The solution to the Liouville equation with an applied rf-field can be simplified under the approximation that the field is applied essentially instantaneous. We will take a limit as the pulse length goes to zero while maintaining a desired pulse angle $\theta_{\boldsymbol{p}}$. Using superscript HP to designate our ideal hard pulse

$$\sigma^{HP} = \lim_{\substack{t_p \to 0 \\ \gamma B_1 t_p \equiv \theta_p}} \sigma^{SP} = \lim_{\substack{t_p \to 0 \\ \gamma B_1 t_p \equiv \theta_p}} [\boldsymbol{U}^{SP}\sigma_0[\boldsymbol{U}^{SP}]^{-1}]$$

$$= \lim_{\substack{t_p \to 0 \\ \gamma B_1 t_p \equiv \theta_p}} [\boldsymbol{U}^{SP}]\sigma_0 \lim_{\substack{t_p \to 0 \\ \gamma B_1 t_p \equiv \theta_p}} [\boldsymbol{U}^{SP}]^{-1}$$

To see how the soft pulse propagator behaves under these imposed conditions, we must look at the effective Hamiltonian in the propagator. We have

$$\lim_{\substack{t_p \to 0 \\ \gamma B_1 t_p \equiv \theta_p}} [\boldsymbol{U}^{SP}] = \lim_{\substack{t_p \to 0 \\ \gamma B_1 t_p \equiv \theta_p}} [e^{\Omega_{rf} F_z t_p} e^{-2\pi i H_{eff} t_p}] = \lim_{\substack{t_p \to 0 \\ \gamma B_1 t_p \equiv \theta_p}} [1 e^{-2\pi i H_{eff} t_p}]$$

Expanding this out proceeds as

$$\lim_{\substack{t_p \to 0 \\ \gamma B_1 t_p \equiv \theta_p}} [\boldsymbol{U}^{SP}] = \lim_{\substack{t_p \to 0 \\ \gamma B_1 t_p \equiv \theta_p}} [e^{-(B_{rf} \boldsymbol{F}_\phi) t_p}] = e^{-\theta_p \boldsymbol{F}_\phi t_p} = \boldsymbol{R}_{xy}(\phi, \theta_p)$$

$$\lim_{\substack{t_p \to 0 \\ \gamma B_1 t_p \equiv \theta_p}} [\boldsymbol{U}^{SP}] = \lim_{\substack{t_p \to 0 \\ \gamma B_1 t_p \equiv \theta_p}} [e^{-2\pi i H_{eff} t_p}] = \lim_{\substack{t_p \to 0 \\ \gamma B_1 t_p \equiv \theta_p}} [e^{-(\boldsymbol{H}_o - \Omega_{rf} \boldsymbol{F}_z + B_{rf} \boldsymbol{F}_\phi) t_p}]$$

Substituting this result into equation xxxx produces

$$\sigma^{HP} = \lim_{\substack{t_p \to 0 \\ \gamma B_1 t_p \equiv \theta_p}} [\boldsymbol{U}^{SP}] \sigma_0 \lim_{\substack{t_p \to 0 \\ \gamma B_1 t_p \equiv \theta_p}} [\boldsymbol{U}^{SP}]^{-1} = \boldsymbol{R}_{xy}(\phi, \theta_p) \sigma_0 [\boldsymbol{R}_{xy}(\phi, \theta_p)]^{-1} \qquad ,$$

and we obtain the following result for propagation of the density matrix through an “ideal” hard pulse.

$$\sigma^{HP} = \boldsymbol{R}_{xy}(\phi, \beta) \sigma_0 [\boldsymbol{R}_{xy}(\phi, \beta)]^{-1} \tag{9-5}$$

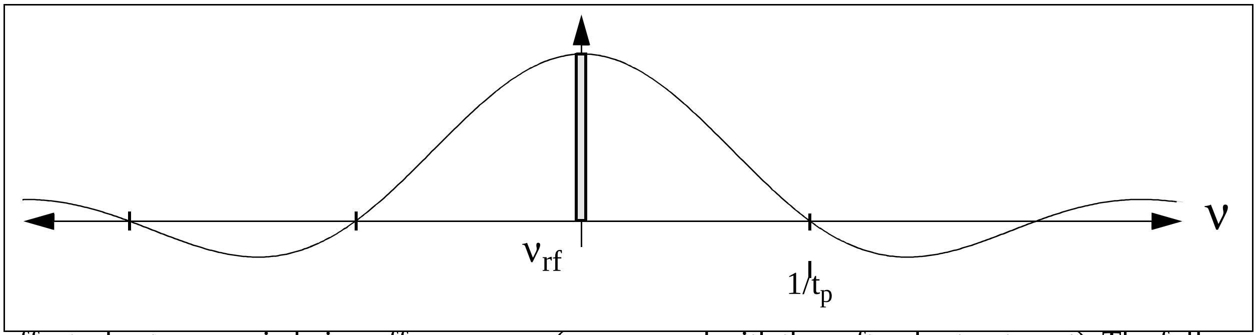
or equivalently

$$\sigma^{IP} = \boldsymbol{U}^{IP} \sigma_0 [\boldsymbol{U}^{IP}]^{-1} \quad \text{with the ideal pulse propagator given by} \quad \boldsymbol{U}^{IP} = \boldsymbol{R}_{xy}(\phi, \beta) \tag{9-6}$$

The physical picture for the “ideal” pulse is that all spins affected by the applied field feel the same field amplitude. That is, the pulse length is short enough that the spins act as if they are all essentially on resonance, i.e. they lie in a bandwidth that is very small compared to the bandwidth contained in the pulse. This is shown in the following diagram with all spin Larmor frequencies contained in the shaded box.

*Figure 19-1* asfasfdasdf

Furthermore, since the ideal pulse is taken to occur instantaneously there are no relative phasing



effects due to any spin being off-resonance (as occurred with the soft pulse treatment). The following diagram compares how three spins, one on resonance and two off-resonance, are affected by a a soft *versus* an ideal π/2 pulse.

## 9.7 Source Codes

```
/* Ipulspin.cc ***************************************-*-c++-*-
**                                                      **
**      Example Program for the GAMMA Library                   **
**                                                              **
** This program tests the ideal pulse function Ixpuls which**
** can be used to selectively pulse a specific spin in an**
** input spin system. The program runs interactively and will**
** perform the following sequence                       **
**                                                      **
**        ________ ________                             **
**       |      | |      |                              **
**       |      | |      |                              **
**       |      | |      |                              **
**       |180   | | 90x  | <---- Acquisition ---->      **
**       | x,i  | |      |                              **
**      ______|      |_|      |_______________________  **
**                                                      **
**                                                      **
** where i is a specific spin in the system. The sequence**
** is repeated for all spins in the system and the 1-D plots**
** output in FrameMaker format. Only homonuclear systems.**
**                                                      **
*************************************************************/

#include <gamma.h>
main(int argc, char* argv[])
{
//               DEFINE SYSTEM & NMR PARAMETERS
//
 String filename;                       // Name of spin system file
 query_parameter(argc, argv, 1,         // Get filename from command
       “\nSpin System Filename? “, filename);// line or ask for it
 spin_system sys;                       // Declare spin system sys
 sys.read(filename);                    // Read system from filename

//            SET OFFSET, SPECTRAL WIDTH, SIZE

 double offset = sys.center();          // Find approx. spectrum center
 sys.offsetShifts(offset);              // Offset shifts so centered
 double NyqF = sys.Nyquist(0, 1.4);     // Approximate Nyquist frequency
 double dt2 = 1.0/(2.0*NyqF);           // t2 time increment
 int t2pts;
 query_parameter(argc, argv, 2,         // FID size
 “\nAcquisition Size? “, t2pts);
//
//               SET UP NECESSARY VARIABLES
//
 block_1D t2BLK(t2pts);                  // 1D-data block storage
 int nspins = sys.spins();
 row_vector plots[nspins];
 gen_op H = Hcs(sys) + HJw(sys);         // Hamiltonian, weak coupling
 gen_op sigma0, sigma1, sigma2;          // Working density matrices
 gen_op D = Fm(sys);                     // F- detect. op.: 0 phase
//
//       APPLY PULSE SEQUENCE, ONCE FOR EACH SPIN
//
 sigma0 = sigma_eq(sys);                 // Equilibirum density matrix
 for(int i=0; i<nspins; i++)
   {
   sigma1 = Ixpuls(sys, sigma0, i, 180.0);// Apply pi x-pulse on spin 2
   sigma2 = Iypuls(sys, sigma1, 90.0);  // Apply second pulse, 90y
   FID(sigma2, D, H, dt2, t2pts, t2BLK);// Acquisition
   exponential_multiply(t2BLK, -8);     // Apodize the FID
   plots[i] = FFT(t2BLK);               // Fourier transform the FID
   }
 double lf, rt;
 lf = offset - NyqF;                     // Set plot limits in Hertz
 rt = offset + NyqF;
 FM_1Dm("plots.mif",nspins,plots,14,5,lf,rt); // Write spectra to a FM
file
}
```

# 10 Rectangular Pulses

## 10.1 Overview

This group of functions facilitate the application of rectangular pulses to density operators. The code is implemented for spin systems containing any number of spins having any I quantum numbers.

## 10.2 Rectangular Pulse Functions

### General Functions

Sxpuls, Sxpuls_U- Pulse/Pulse propagator along the x-axis page 290
Sypuls - Pulse/Pulse propagator along the y-axis page 291
Sxypuls - Pulse/Pulse propagator in the xy-plane page 293

## 10.3 Rectangular Pulse Figures

Approximate Pulse Field Strength page 306
Approximate Pulse Power Spectrum page 306
rf-Field Offset Effects page 307
rf-Field Offset Effects page 307
xxxx xx

## 10.4 Rectangular Pulse Theory

The Liouville Equation page 301
The Rotating Frame page 301

## 10.5 Rectangular Pulse Example Programs

SPIRAL - Figure 22-3xxx page 310
OFFSET - Figure 22-5x page 310

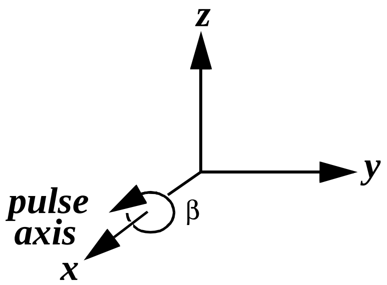
# 10.6 Routines

## 10.6.1 Sxpuls, Sxpuls_U

### Usage:

```
#include <gamma.h >
gen_op Sxpuls(spin_system sys, gen_op sigma, gen_op H, String iso, double offset, double time,
                                                                                    double beta)
gen_op Sxpuls_U(spin_system sys, gen_op H, String iso, double offset, double time, double beta)
```

### Description:



The function either returns a propagator for a rectangular pulse or evolves the input density operator through such a pulse. The pulse is applied to the system ***sys*** which evolves under the static Hamiltonian ***H*** (when the pulse is off) for a length ***time*** on the x-axis at a frequency ***offset*** from the carrier frequency of the isotope type specified, ***iso***. The pulse angle (at the resonance frequency ***offset***) is specified directly by the angle ***beta.*** Note that magnetization along +z will (initially) rotate toward -y when this pulse is applied.The rf-field strength is automatically adjusted to produce the rotation ***beta*** for a spin on resonance.

1. Sxpuls(spin_system &sys, gen_op &sigma, gen_op &H, char* iso, double offset, double time, double beta) Returns a new density operator representing ***sigma*** after the pulse has been applied.
2. Sxpuls_U(spin_system &sys, gen_op &H, char* iso, double offset, double time, double beta) - Returns a propagator for a rectangular pulse with a rotation of angle ***beta*** about the x-axis.

The pulse angle ***beta*** is specified in degrees and has a default value of 90 if left out of the argument list. The pulse length ***time*** is given in seconds and has a default value of 10 msec if both it and the angle are left out of the argument list. The pulse frequency ***offset*** is given in Hertz and has a default value of 0 if it and the subsequent arguments are not specified when calling the function.

### Return Value:

The function returns a general operator.

### Examples:

```
#include <gamma.h >
gen_op sigma, H, U;                               // Set up operators for density matrix, Hamilt. & prop.
spin_system AMX(3);                               // Set up a three spin system.
AMX.isotope(0, “1H”);                             // Set the first spin to a proton.
AMX.isotope(1, “19F”);                            // Set the second spin to a fluorine.
AMX.isotope(2, “19F”);                            // Set the third spin to a fluorine.
sigma = sigma_eq(AMX);                            // Set the density matrix to equilibrium for AMX
H = Ho(AMX);                                      // Set the Hamiltonian to isotropic liquid w/ weak J het.
sigma = Sxpuls(AMX,sigma,H,“19F”,100., 1.e-5,90.); // 90 Pulse on fluorine for 10 usec., 100 Hz
U = Sxpuls(AMX, H, “19F”, 100.0);                 // Propagator for previous pulse. Used defaults.
```

### Mathematical Basis:

The function Sxpuls returns the density matrix propagated through a soft pulse on the x-axis via the formula

$$\sigma^{SP} = \boldsymbol{U}^{SP}\sigma_0[\boldsymbol{U}^{SP}]^{-1}$$

where the propagator is that which is returned by the function Sxpuls_U. In turn, this propagator is given by

$$\boldsymbol{U}^{SP} = \boldsymbol{R}_{\{i\},z}(\theta_{rf})[e^{2\pi i H_{eff} t}]$$

with $\theta_{rf}$ being the angle in degrees that the frame rotating at applied field frequency, $\Omega_{rf}$ (or "freq"), moves during the time, $t_p$ (or "time"), it is applied.

$$\theta_{rf} = \Omega_{rf} t_p$$

The effective Hamiltonian (static in the rotating frame of the rf-field) is given, in Hertz, by

$$\boldsymbol{H}_{eff} = \boldsymbol{H}_0 + \Omega_{rf}\boldsymbol{F}_{\{i\},z} - \left[\frac{\theta_p}{(360 \times t_p)}\right]\boldsymbol{F}_{\{i\},x}$$

The operator $\boldsymbol{H}_0$ (or "H") is the currently active component of the total Hamiltonian without the field, $\theta_p$ (or "theta") is the pulse angle regardless of the pulse length, and the two angular momentum operators, $\boldsymbol{F}_{\{i\},z}$ and $\boldsymbol{F}_{\{i\},x}$ , are obtained from summing over the associated single spin operators for all spins in the spin system "sys" of isotope type "iso". The same spin selectivity is used to compute the z-rotation operator $\boldsymbol{R}_{\{i\},z}$ .

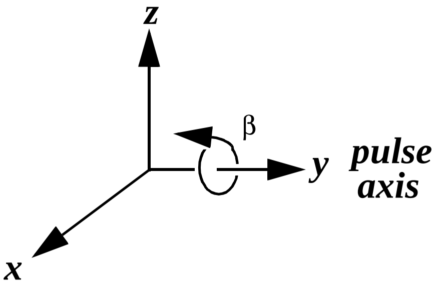
**See Also: SxpulB, Sypul, Sxypul, Ixpuls, Ixpuls_U**

### 10.6.2 Sypuls

**Usage:**

```
#include <gamma.h >
gen_op Sypuls (spin_system sys, gen_op sigma, gen_op H,
                                        String iso, double offset, double time, double beta)
gen_op Sypuls_U (spin_system sys, gen_op H, String iso, double offset, double time, double beta)
```

**Description:**



The function either returns a propagator for a rectangular pulse or evolves the input density operator through such a pulse. The pulse is applied to the system ***sys*** which evolves under the static Hamiltonian ***H*** (when the pulse is off) for a length ***time*** on the y-axis at a frequency ***offset*** from the carrier frequency of the isotope type specified, ***iso***. The pulse angle (at the resonance frequency ***offset***) is specified directly by the angle ***beta.*** Note that magnetization along +z will (initially) rotate toward +x when this pulse is applied.The rf-field strength is automatically adjusted to produce the rotation ***beta*** for a spin on resonance.

1. Sypuls(spin_system &sys, gen_op &sigma, gen_op &H, char* iso, double offset, double time, double beta) - Returns the density matrix operator, sigma, after having been "pulsed" with a rotation of angle beta about the axis. The pulse is applied at a frequency "offset" from the isotope "iso" carrier frequency length "time".

The $B_1$ strength is adjusted to produce the rotation beta for a spin on resonance. Beta is specified in degrees.

2. Sypuls_U(spin_system &sys, gen_op &H, char* iso, double offset, double time, double beta) - Returns the propagator for a soft pulse with a rotation of angle beta about the axis. The pulse is applied at a frequency “offset” from the isotope “iso” carrier frequency length “time”. The $B_1$ strength is adjusted to produce the rotation beta for a spin on resonance. Beta is specified in degrees.

The pulse angle “beta” is specified in degrees and has a default value of 90 if left out of the argument list. The pulse length time is given in seconds and has a default value of 10 msec if both it and the angle are left out of the argument list. The pulse frequency offset is given in Hertz and has a default value of 0 if it and the following to variables are left out of the function argument list.

## Return Value:

The function returns a general operator.

## Example(s):

```
#include <gamma.h >
gen_op sigma, H, U;                          // set up operators for density matrix, Hamilt. & prop.
spin_system AMX(3);                          // set up a three spin system.
AMX.isotope(0, “1H”);                        // set the first spin to a proton.
AMX.isotope(1, “19F”);                       // set the second spin to a fluorine.
AMX.isotope(2, “19F”);                       // set the third spin to a fluorine.
sigma = sigma_eq(AMX);                       // set the density matrix to equilibrium for AMX
H = Ho(AMX);                                 // set the Hamiltonian to isotropic liquid w/ weak J het.
sigma = Sypuls(AMX, sigma, H, “19F”, 100.0,  // 90 Pulse on fluorine for 10 usec., 100 Hz.
                         10.0e-6, 90.0);
U = Sxpuls(AMX, H, “19F”, 100.0);            // Propagator for previous pulse. Used defaults.
```

## Mathematical Basis:

When the function Sypuls returns the propagated density matrix, the formula used is

$$\sigma^{SP} = \boldsymbol{U}^{SP}\sigma_0[\boldsymbol{U}^{SP}]^{-1}$$

where the propagator is that which is returned by the function Sypuls_U. In turn, this propagator is given by

$$\boldsymbol{U}^{SP} = \boldsymbol{R}_{\{i\},z}(\theta_{rf})[e^{2\pi i H_{eff}t}]$$

with $\theta_{rf}$ being the angle in degrees that the frame rotating at applied field frequency, $\Omega_{rf}$ (or “freq”) moves during the time, $t_p$ (or “time”) it is applied.

$$\theta_{rf} = \Omega_{rf}t_p$$

The effective Hamiltonian (static in the rotating frame of the rf-field) is given by

$$\boldsymbol{H}_{eff} = \boldsymbol{H}_0 + \Omega_{rf}\boldsymbol{F}_{\{i\},z} - \left[\frac{\theta_p}{(360 \times t_p)}\right]\boldsymbol{F}_{\{i\},y}$$

Where $\boldsymbol{H}_0$ (or “H”) is the currently active component of the total Hamiltonian without the field, $\theta_p$ (or “theta”) is the pulse angle regardless of the pulse length, and the two angular momentum operators, $\boldsymbol{F}_{\{i\},z}$ and $\boldsymbol{F}_{\{i\},y}$ , are obtained from summing over the associated single spin operators for all spins in the spin system "sys" of isotope type "iso". The same spin selectivity is used to compute the z-rotation operator $\boldsymbol{R}_{\{i\},z}$ .

**See Also: Sypuls, Sxpuls, Sxypul, Iypuls, Iypuls_U**

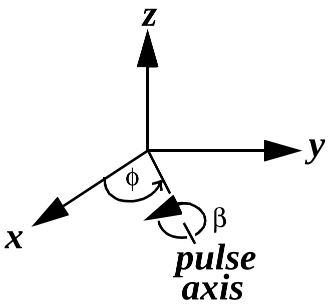
## 10.6.3 Sxypuls

### Usage:

```
#include <gamma.h >
gen_op Sxypuls (spin_system sys, gen_op sigma, gen_op H,
                              String iso, double offset, double time, double beta, double phi)
gen_op Sxypuls_U (spin_system &sys, gen_op &H,
                              char* iso, double offset, double time, double beta, double phi)
gen_op Sxypuls (spin_system &sys, gen_op sigma, gen_op &H, char *iso1, double offset1,
                              char* iso2, double offset2, double time, double beta, double phi)
gen_op Sxypuls_U (spin_system &sys, gen_op &H, char *iso1, double offset1,
                              char* iso2, double offset2, double time, double beta, double phi)
```

### Description:

Applies a "soft pulse" to the density matrix of angle beta about an axis in the xy-plane phi degrees over from the x-axis affecting the spins of the isotope type(s) specified.



1. Sxypuls(spin_system &sys, gen_op &sigma, gen_op &H, char* iso, double offset, double time, double beta, double phi) - Returns the density matrix operator, sigma, after having been "pulsed" with a rotation of angle beta about the axis phi degrees over from the x-axis. The pulse is applied at a frequency "offset" from the isotope "iso" carrier frequency length "time". The $B_1$ strength is adjusted to produce the rotation beta for a spin on resonance.
2. Sxypuls_U(spin_system &sys, gen_op &H, char* iso, double offset, double time, double beta, double phi) - Returns the propagator for a soft pulse with a rotation of angle beta about the axis phi degrees from the x-axis. The pulse is applied at a frequency "offset" from the isotope "iso" carrier frequency length "time". The $B_1$ strength is adjusted to produce the rotation beta for a spin on resonance
3. Sxypuls(spin_system &sys, gen_op &sigma, gen_op &H, char* iso1, double offset1, char* iso2, double offset2, double time, double beta, double phi) - Returns the density matrix operator, sigma, after having been "pulsed" with a rotation of angle beta about the axis phi degrees over from the x-axis.The pulse is simultaneously applied at a frequency "offset1" from the isotope "iso1" carrier frequency and at frequency "offset2" from the isotope "iso2" carrier frequency for a length "time". The $B_1$ strength is adjusted to produce the rotation beta for a spin on the two resonances.
4. Sxypuls_U(spin_system &sys, gen_op &H, char* iso1, double offset1, char* iso2, double offset2, double time, double beta, double phi) - Returns the propagator for a soft pulse with a rotation of angle beta about the axis phi degrees over from the x-axis. The pulse is simultaneously applied at a frequency "offset1" from the

isotope “iso1” carrier frequency and at frequency “offset2” from the isotope “iso2” carrier frequency for a length “time”. The $B_1$ strength is adjusted to produce the rotation beta for a spin on the two resonances.

## Return Value:

The function returns a general operator.

## Example(s):

```
#include <gamma.h >
gen_op sigma, H, U;                                // set up operators for density matrix, Hamilt. & prop.
spin_system AMX(3);                                // set up a three spin system.
AMX.isotope(0, “1H”);                              // set the first spin to a proton.
AMX.isotope(1, “19F”);                             // set the second spin to a fluorine.
AMX.isotope(2, “19F”);                             // set the third spin to a fluorine.
sigma = sigma_eq(AMX);                             // set the density matrix to equilibrium for AMX
H = Ho(AMX);                                       // set the Hamiltonian to isotropic liquid w/ weak J het.
sigma = Sypuls(AMX, sigma, H, “19F”, 100.0,        // 90 Pulse on fluorine for 10 usec., 100 Hz.
                          10.0e-6, 90.0);
U = Sxpuls(AMX, H, “19F”, 100.0);                  // Propagator for previous pulse. Used defaults.
sigma = Sypuls(AMX, sigma, H, “19F”, 100.0,        //
                    “1H”, 10.0, 10.0e-6, 90.0);
U = Sxpuls(AMX, H, “19F”, 100.0, “1H”, 10.0);      // Propagator for previous pulse. Used defaults.
```

## Mathematical Basis:

When the function Sxypuls returns the propagated density matrix, the formula used is

$$\sigma^{SP} = \boldsymbol{U}^{SP}\sigma_0[\boldsymbol{U}^{SP}]^{-1}$$

where the propagator is that which is returned by the function Sxypuls_U. In turn, this propagator is given by either

$$\boldsymbol{U}^{SP} = \boldsymbol{R}_{\{i\},z}(\theta_{rf})[e^{2\pi i H_{eff} t}] \qquad \text{or} \qquad \boldsymbol{U}^{SP} = \boldsymbol{R}_{\{i\},z}(\theta_{i,rf})\boldsymbol{R}_{\{j\},z}(\theta_{j,rf})[e^{2\pi i H_{eff} t}]$$

depending upon whether the field is applied at a single frequency or at two frequencies simultaneously.

For the former (single isotope) case, $\theta_{rf}$ is the angle in degrees that the frame rotating at applied field frequency, $\Omega_{rf}$ (or “freq”) moves during the time, $t_p$ (or “time”) it is applied.

$$\theta_{rf} = \Omega_{rf}t_p$$

The effective Hamiltonian (static in the rotating frame of the rf-field) is given by

$$\boldsymbol{H}_{eff} = \boldsymbol{H}_0 + \Omega_{rf}\boldsymbol{F}_{\{i\},z} - \left[\frac{\theta_p}{(360 \times t_p)}\right]\boldsymbol{F}_{\{i\},xy}(\varphi)$$

Where $\boldsymbol{H}_0$ (or “H”) is the currently active component of the total Hamiltonian without the field, $\theta_p$ (or “theta”) is the pulse angle regardless of the pulse length, and the two angular momentum operators, $\boldsymbol{F}_{\{i\},z}$ and $\boldsymbol{F}_{\{i\},xy}(\varphi)$ , are obtained from summing over the associated single spin operators for all spins in the spin system “sys” of isotope type “iso”. The same spin selectivity is used to compute the z-rotation operator $\boldsymbol{R}_{\{i\},z}$ .

The angle $\varphi$ is the phase angle “phi” given in the argument list.

For the case when two isotopes are affected at the same time, $\theta_{i,\,rf}$ is the angle in degrees that the frame rotating at one applied field frequency, $\Omega_{i,\,rf}$ (or “freq1”) moves during the time, $t_p$ (or “time”) the fields are applied.

$$\theta_{i,\,rf} = \Omega_{i,\,rf} t_p$$

The second frame rotates $\theta_{i,\,rf}$ in an analogous fashion as dictated by $\theta_{j,\,rf} = \Omega_{j,\,rf} t_p$ (and freq2). The effective Hamiltonian (static in the rotating frame of the rf-fields) is given by

$$\boldsymbol{H}_{eff} = \boldsymbol{H}_0 + \Omega_{i,\,rf}\boldsymbol{F}_{\{i\},\,z} + \Omega_{j,\,rf}\boldsymbol{F}_{\{j\},\,z} - \left[\frac{\theta_p}{(360 \times t_p)}\right]\boldsymbol{F}_{\{i+j\},\,xy}(\varphi)$$

Both fields are applied for the same length of time and rotate (at the respective resonWhere $\boldsymbol{H}_0$ (or “H”) is the currently active component of the total Hamiltonian without the field, $\theta_p$ (or “theta”) is the pulse angle regardless of the pulse length, and the two angular momentum operators, $\boldsymbol{F}_{\{i\},\,z}$ and $\boldsymbol{F}_{\{i\},\,xy}(\varphi)$ , are obtained from summing over the associated single spin operators for all spins in the spin system “sys” of isotope type “iso”. The same spin selectivity is used to compute the z-rotation operator $\boldsymbol{R}_{\{i\},\,z}$ . The angle $\varphi$ is the phase angle “phi” given in the argument list.

**See Also: SxypulB, Sxpul, Sypul, Ixypuls, Ixypuls_U**

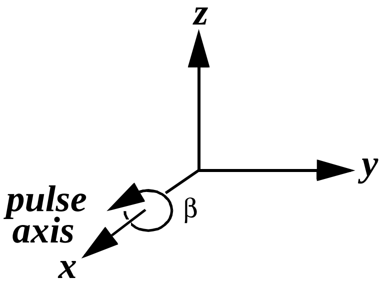
## 10.6.4 SxpulsB

**Usage:**

```
#include <gamma.h >
gen_op SxpulsB (spin_system &sys, gen_op sigma, gen_op &H,
                                        char* iso, double offset, double time, double gamB1)
gen_op SxpulsB_U (spin_system &sys, gen_op &H,
                                        char* iso, double offset, double time, double gamB1)
```

**Description:**

Applies a “soft pulse” to the density matrix of angle $\gamma B_1 t_p$ about the x-axis affecting the spins of the isotope type specified in the argument.



1. SxpulsB(spin_system &sys, gen_op &sigma, char* iso, double offset, double time, double gamB1) - Returns the density matrix operator, sigma, after having been "pulsed" with a rotation of angle beta about the axis. The pulse is applied at a frequency "offset" from the isotope "iso" carrier frequency length "time".
2. SxpulsB_U(spin_system &sys, gen_op &H, char* iso, double offset, double time, double gamB1) - Returns the propagator for a soft pulse with a rotation of angle beta about the axis. The pulse is applied at a frequency "offset" from the isotope "iso" carrier frequency length "time".

## Return Value:

The function returns a general operator.

## Example(s):

```
#include <gamma.h >
gen_op sigma, H, U;                              // set up operators for density matrix, Hamilt. & prop.
spin_system AMX(3);                              // set up a three spin system.
AMX.isotope(0, “1H”);                            // set the first spin to a proton.
AMX.isotope(1, “19F”);                           // set the second spin to a fluorine.
AMX.isotope(2, “19F”);                           // set the third spin to a fluorine.
sigma = sigma_eq(AMX);                           // set the density matrix to equilibrium for AMX
H = Ho(AMX);                                     // set the Hamiltonian to isotropic liquid w/ weak J het.
sigma = Sypuls(AMX, sigma, H, “19F”, 100.0, 10.0e-6, 90.0); // 90 Pulse on fluorine for 10 usec., 100 Hz.
U = Sxpuls(AMX, H, “19F”, 100.0);                // Propagator for previous pulse. Used defaults.
sigma = Sypuls(AMX, sigma, H, “19F”, 100.0, “1H”, 10.0, 10.0e-6, 90.0); // ., 100 Hz.
U = Sxpuls(AMX, H, “19F”, 100.0, “1H”, 10.0);    // Propagator for previous pulse. Used defaults.
```

## Mathematical Basis:

When the function SxpulsB returns the propagated density matrix, the formula used is

$$\sigma^{SP} = \boldsymbol{U}^{SP}\sigma_0[\boldsymbol{U}^{SP}]^{-1}$$

where the propagator is that which is returned by the function SxpulsB_U. In turn, this propagator is given by

$$\boldsymbol{U}^{SP} = \boldsymbol{R}_{\{i\},z}(\theta_{rf})[e^{2\pi i H_{eff} t}]$$

with $\theta_{rf}$ being the angle in degrees that the frame rotating at applied field frequency, $\Omega_{rf}$ (or "freq") moves during the time, $t_p$ (or "time") it is applied.

$$\theta_{rf} = \Omega_{rf} t_p$$

The effective Hamiltonian (static in the rotating frame of the rf-field) is given by

$$\boldsymbol{H}_{eff} = \boldsymbol{H}_0 + \Omega_{rf}\boldsymbol{F}_{\{i\},z} - \gamma B_1 \boldsymbol{F}_{\{i\},x}$$

Where $\boldsymbol{H}_0$ (or "H") is the currently active component of the total Hamiltonian without the field, $\theta_p$ (or "theta") is the pulse angle regardless of the pulse length, and the two angular momentum operators, $\boldsymbol{F}_{\{i\},z}$ and $\boldsymbol{F}_{\{i\},x}$, are obtained from summing over the associated single spin operators for all spins in the spin system "sys" of isotope type "iso". The same spin selectivity is used to compute the z-rotation operator $\boldsymbol{R}_{\{i\},z}$.

**See Also: Sxpul, SypulB, SxypulB, Ixpuls, Ixpuls_U**

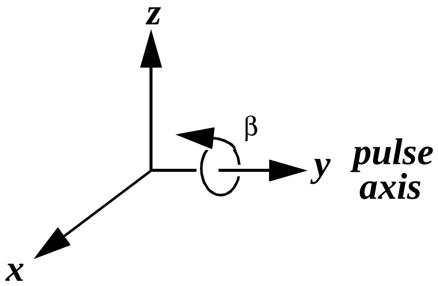
## 10.6.5 SypulsB

### Usage:

```
#include <gamma.h >
gen_op SypulsB (spin_system &sys, gen_op sigma, gen_op &H,
                                        char* iso, double offset, double time, double gamB1)
gen_op SypulsB_U (spin_system &sys, gen_op &H,
                                        char* iso, double offset, double time, double gamB1)
```

### Description:

Applies a "soft pulse" to the density matrix of angle $\gamma B_1 t_p$ about the y-axis affecting the spins of the isotope type specified in the argument.



1. SypulsB(spin_system &sys, gen_op &sigma, char* iso, double offset, double time, double gamB1) - Returns the density matrix operator, sigma, after having been "pulsed" with a rotation of angle beta about the axis. The pulse is applied at a frequency "offset" from the isotope "iso" carrier frequency length "time".
2. SypulsB_U(spin_system &sys, gen_op &H, char* iso, double offset, double time, double gamB1) - Returns the propagator for a soft pulse with a rotation of angle beta about the axis. The pulse is applied at a frequency "offset" from the isotope "iso" carrier frequency length "time".

### Return Value:

The function returns a general operator.

### Example(s):

```
#include <gamma.h >
gen_op sigma, H, U;                 // set up operators for density matrix, Hamilt. & prop.
spin_system AMX(3);                 // set up a three spin system.
AMX.isotope(0, "1H");               // set the first spin to a proton.
AMX.isotope(1, "19F");              // set the second spin to a fluorine.
AMX.isotope(2, "19F");              // set the third spin to a fluorine.
```

```
sigma = sigma_eq(AMX);                                    // set the density matrix to equilibrium for AMX
H = Ho(AMX);                                              // set the Hamiltonian to isotropic liquid w/ weak J het.
sigma = Sypuls(AMX, sigma, H, “19F”, 100.0, 10.0e-6, 90.0); // 90 Pulse on fluorine for 10 usec., 100 Hz.
U = Sxpuls(AMX, H, “19F”, 100.0);                         // Propagator for previous pulse. Used defaults.
sigma = Sypuls(AMX, sigma, H, “19F”, 100.0, “1H”, 10.0, 10.0e-6, 90.0); // ., 100 Hz.
U = Sxpuls(AMX, H, “19F”, 100.0, “1H”, 10.0);             // Propagator for previous pulse. Used defaults.
```

**Mathematical Basis:**

When the function SypulsB returns the propagated density matrix, the formula used is

$$\sigma^{SP} = \boldsymbol{U}^{SP}\sigma_0[\boldsymbol{U}^{SP}]^{-1}$$

where the propagator is that which is returned by the function SypulsB_U. In turn, this propagator is given by

$$\boldsymbol{U}^{SP} = \boldsymbol{R}_{\{i\},z}(\theta_{rf})[e^{2\pi i H_{eff} t}]$$

with $\theta_{rf}$ being the angle in degrees that the frame rotating at applied field frequency, $\Omega_{rf}$ (or “freq”) moves during the time, $t_p$ (or “time”) it is applied.

$$\theta_{rf} = \Omega_{rf}t_p$$

The effective Hamiltonian (static in the rotating frame of the rf-field) is given by

$$\boldsymbol{H}_{eff} = \boldsymbol{H}_0 + \Omega_{rf}\boldsymbol{F}_{\{i\},z} - \gamma B_1 \boldsymbol{F}_{\{i\},y}$$

Where $\boldsymbol{H}_0$ (or “H”) is the currently active component of the total Hamiltonian without the field, $\theta_p$ (or “theta”) is the pulse angle regardless of the pulse length, and the two angular momentum operators, $\boldsymbol{F}_{\{i\},z}$ and $\boldsymbol{F}_{\{i\},y}$, are obtained from summing over the associated single spin operators for all spins in the spin system “sys” of isotope type “iso”. The same spin selectivity is used to compute the z-rotation operator $\boldsymbol{R}_{\{i\},z}$.

**See Also: Sypul, SxpulB, SxypulB, Iypuls, Iypuls_U**

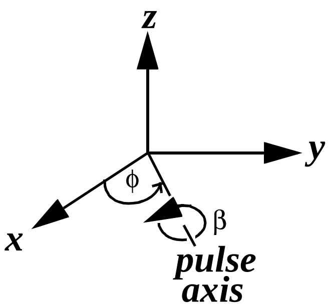
## 10.6.6 SxypulsB

**Usage:**

```
#include <gamma.h >
gen_op SxypulsB (spin_system &sys, gen_op sigma, gen_op &H,
                    char* iso, double offset, double time, double gamB1, double phi)
gen_op SxypulsB_U (spin_system &sys, gen_op &H,
                    char* iso, double offset, double time, double gamB1, double phi)
gen_op SxypulsB (spin_system &sys, gen_op sigma, gen_op &H, char *iso1, double offset1,
                    char* iso2, double offset2, double time, double gamB1, double phi)
gen_op SxypulsB_U (spin_system &sys, gen_op &H, char *iso1, double offset1,
                    char* iso2, double offset2, double time, double gamB1, double phi)
```

## Description:

Applies a “soft pulse” to the density matrix of angle $\gamma B_1 t_p$ about an axis in the xy-plane phi degrees over from the x-axis affecting the spins of the isotope type(s) specified.



1. SxypulsB(spin_system &sys, gen_op &sigma, char* iso, double offset, double time, double gamB1) - Returns the density matrix operator, sigma, after having been “pulsed” with a rotation of angle beta about the axis. The pulse is applied at a frequency “offset” from the isotope “iso” carrier frequency length “time”.
2. SxypulsB_U(spin_system &sys, gen_op &H, char* iso, double offset, double time, double gamB1) - Returns propagator for a soft pulse with a rotation of angle beta about the axis. The pulse is applied at a frequency “offset” from the isotope “iso” carrier frequency length “time”.
3. SxypulsB(spin_system &sys, gen_op &sigma, char* iso, double offset, double time, double gamB1, gen_op &H) - Returns the density matrix operator, sigma, after having been “pulsed” with a rotation of angle beta about the axis. The pulse is applied at a frequency “offset” from the isotope “iso” carrier frequency length “time”.
4. SxypulsB_U(spin_system &sys, gen_op &H, char* iso, double offset, double time, double gamB1) - Returns the propagator for a soft pulse with a rotation of angle beta about the axis. The pulse is applied at a frequency “offset” from the isotope “iso” carrier frequency length “time”.

## Return Value:

The function returns a general operator.

## Example(s):

```
#include <gamma.h >
gen_op sigma, H, U;                          // set up operators for density matrix, Hamilt. & prop.
spin_system AMX(3);                          // set up a three spin system.
AMX.isotope(0, “1H”);                        // set the first spin to a proton.
AMX.isotope(1, “19F”);                       // set the second spin to a fluorine.
AMX.isotope(2, “19F”);                       // set the third spin to a fluorine.
sigma = sigma_eq(AMX);                       // set the density matrix to equilibrium for AMX
H = Ho(AMX);                                 // set the Hamiltonian to isotropic liquid w/ weak J het.
sigma = Sypuls(AMX, sigma, H, “19F”, 100.0, 10.0e-6, 90.0); // 90 Pulse on fluorine for 10 usec., 100 Hz.
U = Sxpuls(AMX, H, “19F”, 100.0);            // Propagator for previous pulse. Used defaults.
sigma = Sypuls(AMX, sigma, H, “19F”, 100.0, “1H”, 10.0, 10.0e-6, 90.0); // ., 100 Hz.
U = Sxpuls(AMX, H, “19F”, 100.0, “1H”, 10.0);  // Propagator for previous pulse. Used defaults.
```

## Mathematical Basis:

When the function Sxypuls returns the propagated density matrix, the formula used is

$$\sigma^{SP} = \boldsymbol{U}^{SP}\sigma_0[\boldsymbol{U}^{SP}]^{-1}$$

where the propagator is that which is returned by the function Sxypuls_U. In turn, this propagator is given by either

$$U^{SP} = R_{\{i\},z}(\theta_{rf})[e^{2\pi i H_{eff} t}] \qquad \text{or} \qquad U^{SP} = R_{\{i\},z}(\theta_{i,rf})R_{\{j\},z}(\theta_{j,rf})[e^{2\pi i H_{eff} t}]$$

depending upon whether the field is applied at a single frequency or at two frequencies simultaneously.

For the former (single isotope) case, $\theta_{rf}$ is the angle in degrees that the frame rotating at applied field frequency, $\Omega_{rf}$ (or “freq”) moves during the time, $t_p$ (or “time”) it is applied.

$$\theta_{rf} = \Omega_{rf} t_p$$

The effective Hamiltonian (static in the rotating frame of the rf-field) is given by

$$H_{eff} = H_0 + \Omega_{rf} F_{\{i\},z} - \left[\frac{\theta_p}{(360 \times t_p)}\right] F_{\{i\},xy}(\varphi)$$

Where $H_0$ (or “H”) is the currently active component of the total Hamiltonian without the field, $\theta_p$ (or “theta”) is the pulse angle regardless of the pulse length, and the two angular momentum operators, $F_{\{i\},z}$ and $F_{\{i\},xy}(\varphi)$, are obtained from summing over the associated single spin operators for all spins in the spin system “sys” of isotope type “iso”. The same spin selectivity is used to compute the z-rotation operator $R_{\{i\},z}$. The angle $\varphi$ is the phase angle “phi” given in the argument list.

For the case when two isotopes are affected at the same time, $\theta_{i,rf}$ is the angle in degrees that the frame rotating at one applied field frequency, $\Omega_{i,rf}$ (or “freq1”) moves during the time, $t_p$ (or “time”) the fields are applied.

$$\theta_{i,rf} = \Omega_{i,rf} t_p$$

The second frame rotates $\theta_{i,rf}$ in an analogous fashion as dictated by $\theta_{j,rf} = \Omega_{j,rf} t_p$ (and freq2). The effective Hamiltonian (static in the rotating frame of the rf-fields) is given, in Hertz, by

$$H_{eff} = H_0 + \Omega_{i,rf} F_{\{i\},z} + \Omega_{j,rf} F_{\{j\},z} - \gamma B_1 F_{\{i+j\},xy}(\varphi)$$

Both fields are applied for the same length of time and rotate (at the respective resonWhere $H_0$ (or “H”) is the currently active component of the total Hamiltonian without the field, $\theta_p$ (or “theta”) is the pulse angle regardless of the pulse length, and the two angular momentum operators, $F_{\{i\},z}$ and $F_{\{i\},xy}(\varphi)$, are obtained from summing over the associated single spin operators for all spins in the spin system “sys” of isotope type “iso”. The same spin selectivity is used to compute the z-rotation operator $R_{\{i\},z}$. The angle $\varphi$ is the phase angle “phi” given in the argument list.

**See Also: Sxypul, SxpulB, SypulB, Ixypuls, Ixypuls_U**

# 10.7 Description

## 10.7.0.1 The Liouville Equation

The Liouville equation or density matrix equation of motion (von Neumann) is[1]

$$ih\frac{d\sigma}{dt} = [\boldsymbol{H},\sigma] \tag{9-7}$$

where $\boldsymbol{H}$ is the acting Hamiltonian. It is easy to verify that the solution to this equation under a ***time independent*** Hamiltonian, $\boldsymbol{H}_0$, is

$$\sigma(t) = e^{-i\boldsymbol{H}_0 t}\sigma_o e^{i\boldsymbol{H}_0 t}. \tag{9-8}$$

Here t is the time during which the Hamiltonian has acted on the spin system, $\sigma_o$ represents the initial state of the system, and $\sigma(t)$ the final state. It is common to write this in terms of the propagator $\boldsymbol{U}$,

$$\sigma(t) = \boldsymbol{U}\sigma_o\boldsymbol{U}^{-1} \qquad \text{where} \qquad \boldsymbol{U} = e^{-i\boldsymbol{H}_0 t} \tag{9-9}$$

## 10.7.0.2 The Rotating Frame

In certain situations, such as under the application of an rf-field, the Hamiltonian is not time independent but has an orderly time dependent component. Although (9-8) is then not applicable, it is feasible under these circumstances that a shift into a rotating coordinate system will make the Hamiltonian appear time independent and some analagous simple equation of motion may be attained. The switch into the rotating frame is accomplished by performing a time-dependent rotation about the z-axis on all operators involved in equation (9-7). Here z is the rotation axis because NMR pulses are normally applied in the xy-plane, perpendicular to the (rotating frame) axis.

$$\tilde{O}p = R_z[\varphi(t)]OpR_z^{-1}[\varphi(t)]. \tag{9-10}$$

A tilde has been used to indicate the operator in the rotating frame.

[1] Our intent in GAMMA is to keep this and all equations in angular frequency units.

# 10.7.0.3 Rotating Frame Liouville Equation

It is easily verified that the equation of motion in the rotating frame is simply

$$ih\frac{d}{dt}\tilde{\sigma} = [\tilde{\boldsymbol{H}} + \Omega F_z, \tilde{\sigma}] , \tag{9-11}$$

highly analogous to (9-7). In light of this transformation, we may now consider application of When an external rf-field to a spin system. The acting Hamiltonian (neglecting relaxation and exchange) is expressed as

$$\boldsymbol{H} = \boldsymbol{H}_0 + \boldsymbol{H}_1(t) ,$$

where $\boldsymbol{H}_0$ is the time independent component (Zeeman, isotropic chemical shift, etc.) and $\boldsymbol{H}_1(t)$ is the time dependent component due to the applied field. If one considers the rf-field to be of angular frequency $\Omega_{rf}$ and phase angle $\phi$ applied along the x-axis, its oscillating magnetic field that can be represented by[1]

$$\vec{\boldsymbol{B}}_{rf}(t) = 2\left|\vec{\boldsymbol{B}}_{rf}\right|\cos(\Omega_{rf}t - \phi)\hat{\boldsymbol{i}} . \tag{9-12}$$

Since, in general, the classical energy of a magnetic dipole interacting with a magnetic field is

$$\boldsymbol{E} = -\vec{\mu} \bullet \vec{\boldsymbol{B}} ,$$

the energy of a dipole moment interacting with the rf-field under consideration is simply

$$\boldsymbol{E}^{rf}(t) = -\vec{\mu} \bullet \vec{\boldsymbol{B}}_{rf}(t) \quad .$$

Replacing $\vec{\mu}$ with $\gamma h\vec{\boldsymbol{I}}$ produces the Hamiltonian for the interaction. For a single spin $i$ and the oscillating field given by (9-12) this is, using now $\boldsymbol{B}_{rf} = \left|\vec{\boldsymbol{B}}_{rf}\right|$

$$\boldsymbol{H}_i^{rf}(t) = -2h\gamma_i\boldsymbol{B}_{rf}\cos(\Omega_{rf}t - \phi)\boldsymbol{I}_{ix} \quad .$$

As is commonly done in the literature[2], both $\boldsymbol{H}_i^{rf}(t)$ (and/or $\boldsymbol{B}_{rf}(t)$) can be broken up into two components having equal magnitudes and angular frequencies but rotating in opposite directions in the xy-plane

$$.\boldsymbol{H}_i^{rf}(t) = -h\gamma_i\boldsymbol{B}_{rf}\boldsymbol{I}_{ix} \bullet [\underbrace{[\cos(\Omega_{rf}t - \phi)\boldsymbol{i} + \sin(\Omega_{rf}t - \phi)\boldsymbol{j}]}_{\text{counter-clockwise}} + \underbrace{[\cos(\Omega_{rf}t - \phi)\boldsymbol{i} - \sin(\Omega_{rf}t - \phi)\boldsymbol{j}]}_{\text{clockwise}}]$$

Here, $\boldsymbol{i}$ and $\boldsymbol{j}$ are unit vectors on the x and y axes respectively. It has been demonstrated that only the component rotating in the direction of the Larmor precession, the clockwise component, will

[1] The factor of 2 in this equation if merely for convenience. It will disappear from successive equations as they are manipulated algebraically. The phase angle $\phi$ allows for pulses about any axis in the xy-plane and maintains complete generality of the equation.

[2] For example, see **********

be effective. Thus to a very good degree of approximation

$$\boldsymbol{H}_i^{rf}(t) = -h\gamma_i \boldsymbol{B}_{rf}\boldsymbol{I}_{ix} \bullet [\cos(\Omega_{rf}t - \phi)\boldsymbol{i} - \sin(\Omega_{rf}t - \phi)\boldsymbol{j}]$$

This is simply the scaled $\boldsymbol{I}_{ix}$ vector rotating about the z-axis and can in fact be rewritten in terms of rotations about z.

$$\boldsymbol{H}_i^{rf}(t) = -(h\gamma_i)\boldsymbol{B}_{rf}\boldsymbol{R}_z^{-1}(\Omega_{rf}t)\boldsymbol{R}_z(\phi)\boldsymbol{I}_{ix}\boldsymbol{R}_z^{-1}(\phi)\boldsymbol{R}_z(\Omega_{rf}t)$$

The total rf-field Hamiltonian for a spin system is obtained by summing over all spins,

$$\boldsymbol{H}_i^{rf}(t) = -h\boldsymbol{B}_{rf}\sum_{i}^{spins}\{\gamma_i\boldsymbol{R}_z^{-1}(\Omega_{rf}t)\boldsymbol{R}_z(\phi)\boldsymbol{I}_{ix}\boldsymbol{R}_z^{-1}(\phi)\boldsymbol{R}_z(\Omega_{rf}t)\}$$

For convenience we can define the total magnetic moment operator as

$$\vec{M} = \sum_{i}^{spins}\gamma_i\vec{\boldsymbol{I}}_i$$

and the phased x-component of this operator as

$$M_{x,\varphi} = \boldsymbol{R}_z(\phi)M_x\boldsymbol{R}_z^{-1}(\phi).$$

Our total rf-field Hamiltonian is then

$$\boldsymbol{H}_i^{rf}(t) = -h\boldsymbol{B}_{rf}\{\boldsymbol{R}_z^{-1}(\Omega_{rf}t)M_{x,\varphi}\boldsymbol{R}_z(\Omega_{rf}t)\} \qquad (9\text{-}13)$$

We now wish to solve the equation of motion in the rotating frame, equation (9-11), under the effects of a static NMR Hamiltonian, $\boldsymbol{H}_o$ and an applied rf-field Hamiltonian, $\boldsymbol{H}_1(t)$ . The applicable equations are all grouped together below for convenience.

*NMR Equations for Application of an RF-Field*

$$ih\frac{d}{dt}\tilde{\sigma} = [\tilde{\boldsymbol{H}} + \Omega F_z, \tilde{\sigma}] = [\boldsymbol{H}_{eff}, \tilde{\sigma}]$$

$$\boldsymbol{H} = \boldsymbol{H}_0 + \boldsymbol{H}_1(t)$$

$$\boldsymbol{H}_i^{rf}(t) = -h\boldsymbol{B}_{rf}\{\boldsymbol{R}_z^{-1}(\Omega_{rf}t)M_{x,\varphi}\boldsymbol{R}_z(\Omega_{rf}t)\}$$

$$\tilde{O}p = R_z(\varphi)OpR_z^{-1}(\varphi)$$

The first step toward a solution is to transform the acting Hamiltonian into a rotating frame.

$$\tilde{\boldsymbol{H}} = \tilde{\boldsymbol{H}}_0 + \tilde{\boldsymbol{H}}_1(t) = \boldsymbol{H}_0 + \tilde{\boldsymbol{H}}_1(t) = \boldsymbol{H}_{eff}$$

We choose the frame to be rotating at a frequency $\Omega_{ref}$, about the z-axis. The isotropic NMR

Hamiltonian, $\boldsymbol{H}_o$, is rotationally invariant and will not change in the rotating frame and then $\boldsymbol{H}_o = \tilde{\boldsymbol{H}}_0$. The rf-field Hamiltonian in the rotating frame is

$$\tilde{\boldsymbol{H}}_1(t) = R_z(\Omega_{ref}t)\boldsymbol{H}_1(t)R_z^{-1}(\Omega_{ref}t)$$

$$\tilde{\boldsymbol{H}}_1(t) = R_z(\Omega_{ref}t)[-h\boldsymbol{B}_{rf}\{\boldsymbol{R}_z^{-1}(\Omega_{rf}t)M_{x,\varphi}\boldsymbol{R}_z(\Omega_{rf}t)\}]R_z^{-1}(\Omega_{ref}t)$$

$$\tilde{\boldsymbol{H}}_1(t) = R_z([\Omega_{ref}-\Omega_{rf}]t)[-h\boldsymbol{B}_{rf}M_{x,\varphi}]R_z^{-1}([\Omega_{ref}-\Omega_{rf}]t)$$

The total Hamiltonian in the rotating frame is then

$$\tilde{\boldsymbol{H}}_1(t) = \boldsymbol{H}_0 + R_z([\Omega_{ref}-\Omega_{rf}]t)[-h\boldsymbol{B}_{rf}M_{x,\varphi}]R_z^{-1}([\Omega_{ref}-\Omega_{rf}]t)$$

and the effective Hamiltonian

$$\boldsymbol{H}_{eff} = \boldsymbol{H}_{eff}(t) = \boldsymbol{H}_0 + \Omega_{ref}F_z + R_z([\Omega_{ref}-\Omega_{rf}]t)[-h\boldsymbol{B}_{rf}M_{x,\varphi}]R_z^{-1}([\Omega_{ref}-\Omega_{rf}]t)$$

Since we desire the effective Hamiltonian to appear time independent to facilitate the solution to the equation of motion, ***we must choose our rotating reference frame to be identical to that of the applied rf-field***. In other words, it is necessary that we pick $\Omega_{ref}$ such that $\Omega_{ref} = \Omega_{rf}$. When this is done it is easy to see how the effective Hamiltonian appears time independent.

$$\boldsymbol{H}_{eff}\Big|_{\Omega = \Omega_{ref} = \Omega_{rf}} = \boldsymbol{H}_o + \Omega\boldsymbol{F}_z - hB_{rf}M_{x,\varphi}. \tag{9-14}$$

Note that for a homonuclear spin system this is

$$\boldsymbol{H}_{eff} = \boldsymbol{H}_o + \Omega_{rf}\boldsymbol{F}_z - h\boldsymbol{B}_{rf}\boldsymbol{F}_{x,\varphi}$$

where $\boldsymbol{F}_{x,\varphi} = \boldsymbol{R}_z(\phi)\boldsymbol{F}_x\boldsymbol{R}_z^{-1}(\phi)$. For a single spin with the rf-field on resonance (the field frequency exactly matches the spin Larmor frequency) two of the terms in equation exactly cancel, $\boldsymbol{H}_o - \Omega_{rf}\boldsymbol{F}_z = 0$ leaving only $\boldsymbol{H}_{eff} = B_{rf}\boldsymbol{F}_\phi$ and the effective propagator is of course seen to be identical to that of an ideal pulse (see the chapter Ideal Pulses in this document).

Since the effective Hamiltonian appears time independent in the frame rotating with the applied field, the previous solution to the Liouville equation, (9-8), can now be utilized.

$$\tilde{\sigma}^P_{rfrot} = e^{-2\pi i H_{eff}t_p}\tilde{\sigma}_{0,\,rfrot}e^{2\pi i H_{eff}t_p} \tag{9-15}$$

We use the superscript $P$ to indicate the density matrix after a pulse and $t_p$ for the time the pulse was applied. The tilde is used to indicate that the operator is in the rotating frame and the specific frame is synchronous with the rf-field frequency as indicated by the subscript $rfrot$. The final form of the solution is obtained by switching the density matrices out of the rf-field rotating frame back into whatever frame the density matrix was originally in. Applying the inverse of equation (9-10) we have then

$$\sigma^P = e^{\Omega_{rf}F_zt}[e^{2\pi i H_{eff}t_p}]e^{-\Omega_{rf}F_zt_0}\sigma_0e^{\Omega_{rf}F_zt_0}[e^{-2\pi i H_{eff}t_p}]e^{-\Omega_{rf}F_zt}.$$

Here $t_0$ is the initial time, $t$ the final time, and $t_p$ the time the field is on. They relate to each other with $t = t_p + t_0$. As these last transformations are also just rotations about the z-axis by the rf-field phase angle at a specific time, we can define $\theta_{rf} = \Omega_{rf}t$ and $\theta_{rf,0} = \Omega_{rf}t_0$ to make the nomenclature more compact. The final result for propagation of the density matrix through a pulse is

$$\sigma^P = \boldsymbol{R}_z(\theta_{rf})[e^{2\pi i H_{eff}t_p}]\boldsymbol{R}_z^{-1}(\theta_{rf,0})\sigma_0\boldsymbol{R}_z(\theta_{rf})[e^{-2\pi i H_{eff}t_p}]\boldsymbol{R}_z^{-1}(\theta_{rf}), \qquad \text{(9-16)}$$

or equivalently

$$\sigma^P = \boldsymbol{U}^P\sigma_0[\boldsymbol{U}^P]^{-1} \qquad \text{(9-17)}$$

with the pulse propagator given by

$$\boldsymbol{U}^P = \boldsymbol{R}_z(\theta_{rf})e^{-2\pi i H_{eff}t_p}\boldsymbol{R}_z^{-1}(\theta_{rf,0}) \qquad \text{(9-18)}$$

where

$$\theta_{rf} = \Omega_{rf}t, \qquad \theta_{rf,0} = \Omega_{rf}t_0, \qquad \text{and} \qquad \boldsymbol{H}_{eff} = \boldsymbol{H}_o + \Omega_{rf}\boldsymbol{F}_z - hB_{rf}M_{x,\varphi}. \qquad \text{(9-19)}$$

.

*NMR Equations for RF Pulses*

$$\sigma^P = \boldsymbol{U}^P\sigma_0[\boldsymbol{U}^P]^{-1}$$

$$\boldsymbol{U}^P = \boldsymbol{R}_z(\theta_{rf})e^{-2\pi i H_{eff}t_p}\boldsymbol{R}_z^{-1}(\theta_{rf,0})$$

$$\boldsymbol{H}_{eff} = \boldsymbol{H}_o + \Omega_{rf}\boldsymbol{F}_z - hB_{rf}M_{x,\varphi}$$

We consider the mathematical description of the application of two successive pulses to a spin system. The first pulse (P1) is given by

$$\sigma^{P1} = \boldsymbol{U}^{P1}\sigma_0[\boldsymbol{U}^{P1}]^{-1}$$

ant the second (P2) by

$$\sigma^{P2} = \boldsymbol{U}^{P2}\sigma^{P1}[\boldsymbol{U}^{P2}]^{-1} = \boldsymbol{U}^{P2}\boldsymbol{U}^{P1}\sigma_0[\boldsymbol{U}^{P1}]^{-1}[\boldsymbol{U}^{P2}]^{-1} \qquad .$$

When this is expanded out it is easy to see how the two pulses are expressed as a single propagator.

$$\boldsymbol{U}^{P2}\boldsymbol{U}^{P1} = [\boldsymbol{R}_z(\theta_{rf,2})e^{-2\pi i H_{eff,2}t_{p2}}\boldsymbol{R}_z^{-1}(\theta_{rf,1})][\boldsymbol{R}_z(\theta_{rf,1})e^{-2\pi i H_{eff,1}t_{p1}}\boldsymbol{R}_z^{-1}(\theta_{rf,0})]$$

$$\boldsymbol{U}^{P2}\boldsymbol{U}^{P1} = [\boldsymbol{R}_z(\theta_{rf,2})e^{-2\pi i H_{eff,2}t_{p2}}e^{-\Omega_{rf}F_z(t_{p1}+t_0)}][e^{\Omega_{rf}F_z(t_{p1}+t_0)}e^{-2\pi i H_{eff,1}t_{p1}}\boldsymbol{R}_z^{-1}(\theta_{rf,0})]$$

$$\boldsymbol{U}^{P2}\boldsymbol{U}^{P1} = [\boldsymbol{R}_z(\theta_{rf,2})e^{-2\pi i H_{eff,2}t_{p2}}e^{-2\pi i H_{eff,1}t_{p1}}\boldsymbol{R}_z^{-1}(\theta_{rf,0})]$$

The inner rotations will always cancel out so that multiple pulse propagator may always be combined.

Recall that GAMMA normally works in a rotating reference frame (or in a multiple rotating frame) from the outset of an MR simulation, *e.g.* 500 MHz for a proton 500 MHz NMR simulation. By design, GAMMA performs density matrix computations in a rotating frame (or multiple rotating frame in heteronuclear spin systems). This means that for each isotope type in a spin system the program works referenced to axes which are rotating in the direction of the Larmor precession at the Larmor frequency.Thus, we must shift from any default rotating reference frame(s) into the field rotating frame in order for this solution to work properly. We shall shortly return to this point, but for now we have

Were one able to switch an rf-field on and off instantaneously a plot of the field amplitude versus time would be a square wave (as seen from the frame rotating at the frequency of the field itself). This is depicted below for a pulse at length $t_p$ of amplitude $1/t_p$ and beginning at time $-t_p/2$.

***Approximate Pulse Field Strength***

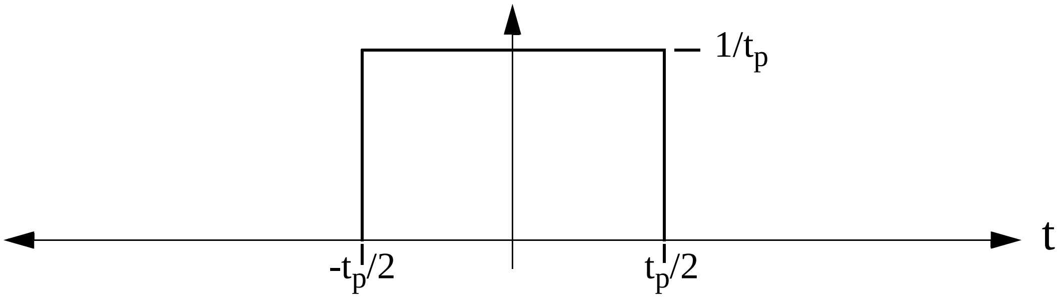


*Figure 22-1* Amplitude of an rf-field which can instantaneously be switched on and off and is of constant amplitude and phase. This is in the frame rotating with the field itself.

Due to the finite time the field is on, the “pulse” contains not only the frequency of the field itself but a multitude of frequencies about this value. By taking the Fourier transform of the time domain spectrum of the pulse one obtains a picture of the frequency spectrum, shown below.

***Approximate Pulse Power Spectrum***

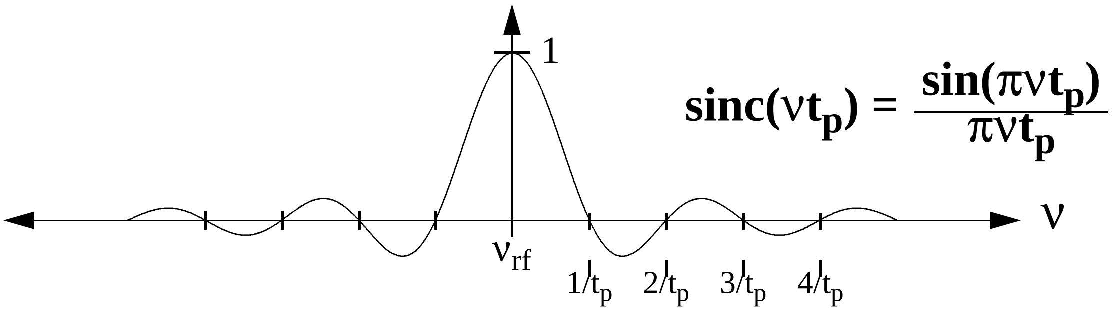


*Figure 22-2* **The Fourier transform of Figure 22-1 approximating the power spectrum of a pulse.**

Of course this is only an approximation, one reason being that the field cannot be instantaneously turned on and off. Another reason is that the pulse is usually applied not at $-t_p/2$ but at time zero and this results in a phase effect placed on the sinc function[1]. Furthermore, the effect on individual spins will vary due the frequency offset between their respective Larmor frequencies and the field frequency. The following plot shows the response of z-magnetization to a p/2 pulse (at zero offset) as this offset is increased[2].

***rf-Field Offset Effects***

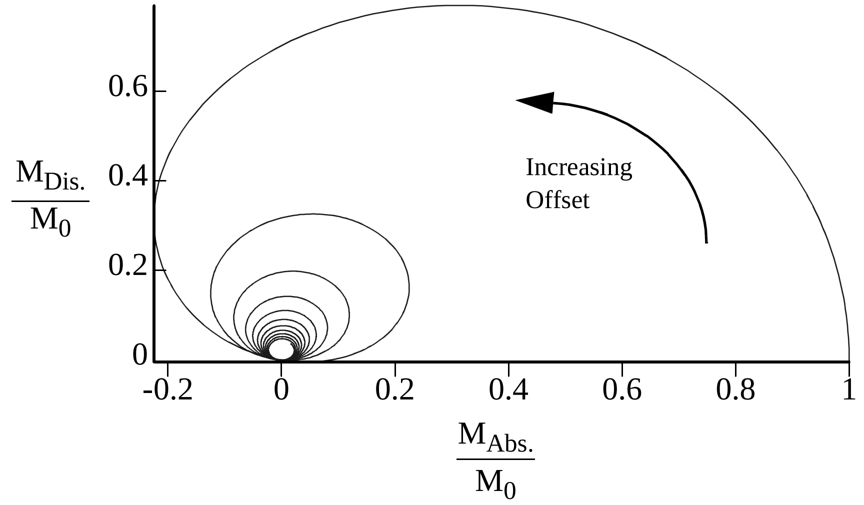


***Figure 22-3*** **The effects of B1 offset on peak intensity and phase.[3]**

Displayed in an alternative fashion, the following figure shows more explicitly on the absorption, dispersion, and magnitude of a transition will vary with frequency offset. Generally the farther away the transition is from the offset, the less effect the applied field has on the spin (associated with the transition).

***rf-Field Offset Effects***

***Figure 22-4*** **The effects of B1 offset on peak intensity in terms of the absorption, dispersion, and magnitude respectively.[4]**

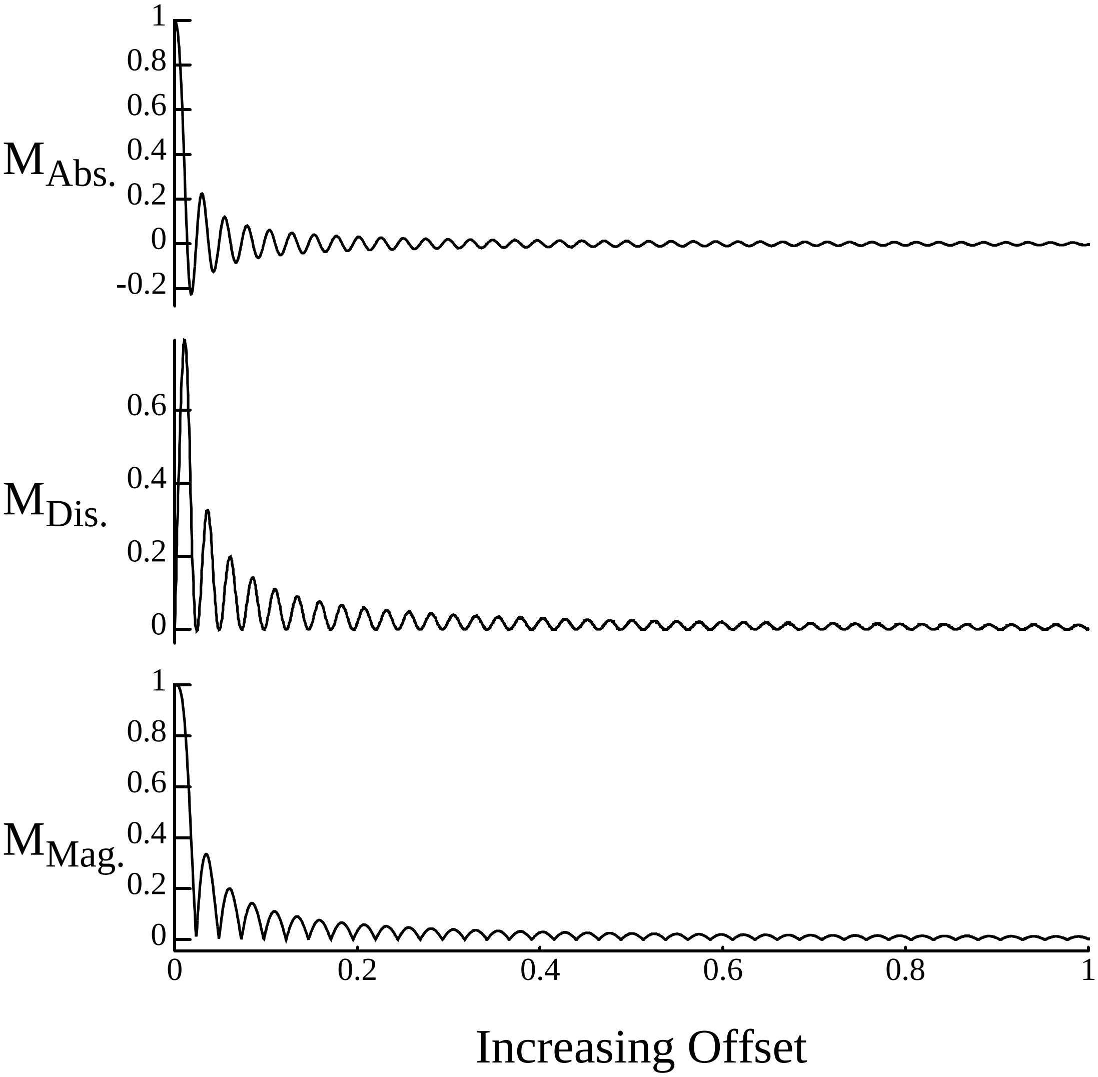
(9-20)

[1] This is a well known property of the Fourier transform. Had the box function started at t = 0 the transformed spectrum would be a complex function, a blend of a real and imaginary sinc functions.

[2] This plot is described in more detail in Fukushima and Roeder, page 56.

[3] The code which generated this figure is given at the end of this chapter as spiral.cc.

[4] The code to generate this figure is not included. It is merely a slight modification of spiral.cc given at the end of this chapter. The first two plots require replacing the function FrameMaker_xyplot with FrameMaker. The last plot requires an additional data block containing the magnitude of the original data block and an additional FrameMaker function call with this data block to produce the plot file.


1
0.8
0.6
$M_{Abs.}$
0.4
0.2
0
-0.2
0.6
0.4
$M_{Dis.}$
0.2
0
1
0.8
0.6
$M_{Mag.}$
0.4
0.2
0
0
0.2
0.4
0.6
0.8
1
Increasing Offset

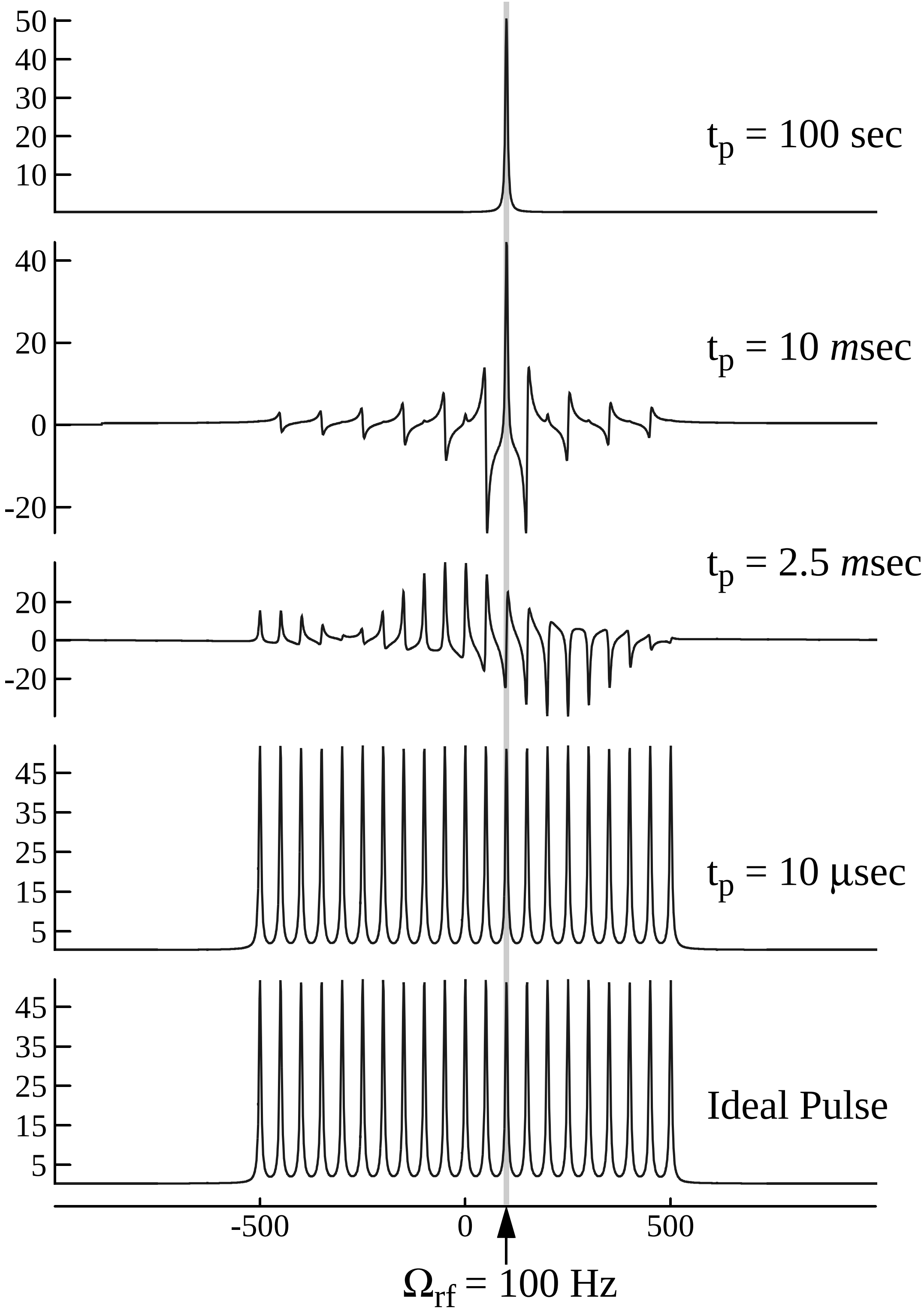


***Figure 22-5*** **The effects of B1 offset on peak intensity and phase.**[1]

## 10.8 Example Source Codes

### SPIRAL - Figure 22-3

```
   /*-*- c++ ------ spiral.cc ------ GAMMA soft pulse example *-*/

#include <gamma.h>
main ()
{
 int npts = 2048;
 block_1D SPECTRA(npts);                          // block for data storage
 spin_system sys(1);                              // single spin system
 double offset = 100.;                            // initial offset at 100
 complex z;
 sys.shift(0,100.);                               // Larmor at 100 also
 gen_op dnsmx;
 gen_op H = Ho(sys);
 gen_op dnseq = sigma_eq(sys);                    // equilibrium dens. mx.
 gen_op detect = Fm(sys, "1H");                   // detector to F-
 for(int i=0; i<npts; i++)
   {
   dnsmx = dnseq;                                 // equilibrium dens. mx.
   dnsmx = Sxpuls(sys, dnsmx, H, "1H",
                 offset, 1.0, 90.);               // apply pulse.
     if(i == 0)
     {
     z = trace(detect, dnsmx);                    // compute Mo from pt. 1
     SPECTRA(i) = 1.0;                            // set first point to 1.0
     }
   else
     SPECTRA(i) = trace(detect,dnsmx)/z;          // compute M
   offset -= .02;                                 // increment offset
   }
 FM_xyPlot("spiral.mif",SPECTRA);                 // output FM plot file
 }
```

### OFFSET - Figure 22-5

```
   /*-*- c++ ------ offset.cc ------ GAMMA soft pulse example *-*/

#include <gamma.h>

main ()
 {
 block_1D SPECTRA1(2048),
                                                SPECTRA2(2048), SPECTRA(2048);
 spin_system ab(1);                               // initial spin system
 double cshift;
 cshift = -550.;
 ab.shift(0,cshift);
 gen_op dnseq, dnsmx;
 dnseq = sigma_eq(ab);
 gen_op H;                                        // initial static H
 H = Ho(ab);
 gen_op detect;                                   // detection operator
 detect = Fm(ab, "1H");
 double time;                                     // input pulse length
 cout << "\nPulse length (time)?\n";
```

[1] The code for the program which generated this figure (run 4 times at the appropriate times with results added) is given at the end of this chapter as offset.cc.

```
 cin >> time;
 for(int i=0; i<21; i++)
 {
 dnsmx = dnseq;                               // reset density matrix
 cshift += 50.;                               // increment shift 50 Hz
 ab.shift(0,cshift);
 H = Ho(ab);                                  // recompute static Ham.
 dnsmx = Sypuls(ab, dnsmx , H,
                                              “1H”, 100., time, 90.);
 FID(dnsmx, detect, H, 0.0005, 2048,
                                              SPECTRA);
 SPECTRA1 += SPECTRA;                         // sum FID
 dnsmx = dnseq;                               // reset density matrix
 dnsmx = Iypuls(ab, dnsmx , 90.);             // analogous ideal pulse
 FID(dnsmx, detect, H, 0.0005, 2048,
                                              SPECTRA);
 SPECTRA2 += SPECTRA;                         // sum FID
 }
 exponential_multiply(SPECTRA1,-20.);         // apodize soft pulse FID
 exponential_multiply(SPECTRA2,-20.);         // apodize hard pulse FID
 SPECTRA1=FFT(SPECTRA1);                      // Fourier transform
 SPECTRA2=FFT(SPECTRA2);                      // Fourier transform
 FM_1D(“sp.mif”, SPECTRA1, 13, 5, -1000, 1000);
 FM_1D(“hp.mif”, SPECTRA2, 13, 5, -1000, 1000);
}
```

# 11 Shaped Pulses

## 11.1 Overview

This module supplies functions which either apply, or return propagators for, shaped pulses. The general functions invariably take a GAMMA row_vector as an input argument to define the shape of the pulse. The row_vector supplies the number if pulse divisions (vector points) and the intensity & phase (point values) at each division. Each division is sconstructed as a “real pulse” either with or without relaxation effects accounted for. Additional functions are supplied for some of the more common pulse shapes in NMR.

## 11.2 Shaped Pulse Functions

**Functions That Apply Pulses**

| | | |
|---|---|---|
| Shxpuls | - Shaped pulses on the x-axis | page 11-312 |
| Iypuls | - Pulses on the y-axis | xxx |
| Ixypuls | - Pulses in the xy-plane | xxx |

**Functions That Return Pulse Propagators**

| | | |
|---|---|---|
| Ixpuls_U | - Propagator for pulse on the x-axis | xxxx |
| Iypuls_U | - Propagator for pulse on the y-axis | xx |
| Ixypuls_U | - Propagator for pulse in the xy-plane | xx |

## 11.3 Routines

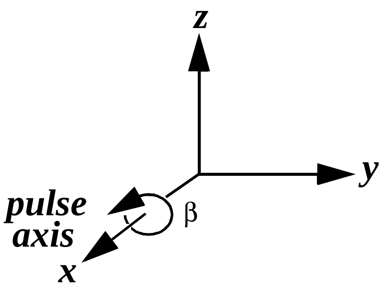
### 11.3.1 Shxpuls

**Usage:**

```
#include <gamma.h >
gen_op Shxpuls (spin_system &sys, d_block &BLK, gen_op sigma, gen_op &H,
                                        char* iso, double offset, double
time, double beta)
gen_op Shxpuls_U (spin_system &sys, d_block &BLK, gen_op &H,
                                        char* iso, double offset, double
time, double beta)
```

**Description:**

The function either returns a propagator for a “shaped pulse” or evolves the input density matrix through this pulse. The pulse is applied for a length “time” on the x-axis at a frequency “offset” from the carrier frequency of the isotope type specified, “iso” . The total pulse angle (at the resonance frequency “offset”) is specified directly by the angle beta. The pulse shape is specified by the real values contained in the data block “BLK”



1. Shxpuls(spin_system &sys, d_block &BLK, gen_op &sigma, gen_op &H, char* iso, double offset, double time, double beta) - Returns the density matrix operator, sigma, after having been “pulsed” from the x-axis with a total rotation of angle beta about the axis. The pulse is applied at a frequency “offset” from the isotope “iso” carrier frequency for a length “time”. The rf-field strength is automatically adjusted to produce the rotation beta for a spin on resonance. The pulse shape is specified by the real values in the data block BLK.
2. Shxpuls_U(spin_system &sys, d_block &BLK, gen_op &H, char* iso, double offset, double time, double beta) - Returns the propagator for a soft pulse with a rotation of angle beta about the axis. The pulse is applied at a frequency “offset” from the isotope “iso” carrier frequency for a length “time”. The rf-field strength is automatically adjusted to produce the rotation beta for a spin on resonance.

The pulse angle “beta” is specified in degrees and has a default value of 90 if left out of the argument list. The pulse length time is given in seconds and has a default value of 10 msec if both it and the angle are left out of the argument list. The pulse frequency offset is given in Hertz and has a default value of 0 if it and the following to variables are left out of the function argument list.

### Return Value:

The function returns a general operator.

### Examples:

```
#include <gamma.h >
gen_op sigma, H, U;                          // set up operators for density matrix, Hamilt. & prop.
d_block BLK(100);                            // set up a three spin system.
spin_system AMX(3);                          // set up a three spin system.
AMX.isotope(0, “1H”);                        // set the first spin to a proton.
AMX.isotope(1, “19F”);                       // set the second spin to a fluroine.
AMX.isotope(2, “19F”);                       // set the third spin to a fluorine.
sigma = sigma_eq(AMX);                       // set the density matrix to equilibrium for AMX
H = Ho(AMX);                                 // set the Hamiltonian to isotropic liquid w/ weak J het.
BLK = Gaussian(100, 50);                     // set the data block to a Gaussian linshape.
sigma = Shxpuls(AMX, BLK, sigma, H, “19F”, 100.0); // 90 Pulse on fluorine for 10 usec., 100 Hz.
U = Shxpuls(AMX, H, “19F”, 100.0);           // Propagator for previous pulse. Used defaults.
```

### Mathematical Basis:

The function Shxpuls returns the density matrix propagated through a shaped pulse on the x-axis via the fomula

$$\sigma^{SHP} = \boldsymbol{U}^{SHP}\sigma_0[\boldsymbol{U}^{SHP}]^{-1}$$

where the propagator is that which is returned by the function Sxpuls_U. In turn, this propagator is produced from the product of many soft pulses as

$$\boldsymbol{U}^{SHP} = \prod_i [\boldsymbol{U}^{SP}]_i \ .$$

**See Also:**

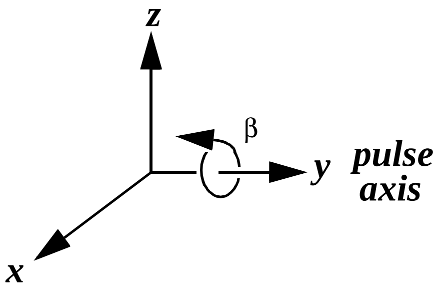
## 11.3.2 Shypuls

**Usage:**

```
#include <gamma.h >
gen_op Shypuls (spin_system &sys, d_block &BLK, gen_op sigma, gen_op &H,
                                        char* iso, double offset, double
time, double beta)
gen_op Shypuls_U (spin_system &sys, d_block &BLK, gen_op &H,
                                        char* iso, double offset, double
time, double beta)
```

**Description:**

The function either returns a propagator for a “shaped pulse” or evolves the input density matrix through this pulse. The pulse is applied for a length “time” on the y-axis at a frequency “offset” from the carrier frequency of the isotope type specified, “iso” . The total pulse angle (at the resonance frequency “offset”) is specified directly by the angle beta. The pulse shape is specified by the real values contained in the data block “BLK”.



1. Shypuls(spin_system &sys, d_block &BLK, gen_op &sigma, gen_op &H, char* iso, double offset, double time, double beta) - Returns the density matrix operator, sigma, after having been “pulsed” from the y-axis with a total rotation of angle beta about the axis. The pulse is applied at a frequency “offset” from the isotope “iso” carrier frequency for a length “time”. The rf-field strength is automatically adjusted to produce the rotation beta for a spin on resonance. The pulse shape is specified by the real values in the data block BLK.
2. Shypuls_U(spin_system &sys, d_block &BLK, gen_op &H, char* iso, double offset, double time, double beta) - Returns the propagator for a soft pulse with a rotation of angle beta about the y axis. The pulse

is applied at a frequency “offset” from the isotope “iso” carrier frequency for a length “time”. The rf-field strength is automatically adjusted to produce the rotation beta for a spin on resonance.

The pulse angle “beta” is specified in degrees and has a default value of 90 if left out of the argument list. The pulse length time is given in seconds and has a default value of 10 msec if both it and the angle are left out of the argument list. The pulse frequency offset is given in Hertz and has a default value of 0 if it and the following to variables are left out of the function argument list.

## Return Value:

The function returns a general operator.

## Example(s):

```
#include <gamma.h >
gen_op sigma, H, U;                          // set up operators for density matrix, Hamilt. & prop.
spin_system AMX(3);                          // set up a three spin system.
AMX.isotope(0, “1H”);                        // set the first spin to a proton.
AMX.isotope(1, “19F”);                       // set the second spin to a fluroine.
AMX.isotope(2, “19F”);                       // set the third spin to a fluorine.
sigma = sigma_eq(AMX);                       // set the density matrix to equilibrium for AMX
H = Ho(AMX);                                 // set the Hamiltonian to isotropic liquid w/ weak J het.
sigma = Shypuls(AMX, BLK, sigma, H, “19F”, 100.0); // 90 Pulse on fluorine for 10 usec., 100 Hz.
U = Shypuls(AMX, H, “19F”, 100.0);           // Propagator for previous pulse. Used defaults.
```

## Mathematical Basis:

The function Shypuls returns the density matrix propagated through a shaped pulse on the yx-axis via the fomula

$$\sigma^{SHP} = \boldsymbol{U}^{SHP}\sigma_0[\boldsymbol{U}^{SHP}]^{-1}$$

where the propagator is that which is returned by the function Shypuls_U. In turn, this propagator is produced from the product of many soft pulses as

$$\boldsymbol{U}^{SHP} = \prod_k [\boldsymbol{U}^{SP}]_k .$$

The soft pulse propagator is that which is returned by the function Sypuls_U. In turn, this propagator is given by

$$\boldsymbol{U}^{SP} = \boldsymbol{R}_{\{i\},z}(\theta_{rf})[e^{2\pi i H_{eff} t}]$$

with $\theta_{rf}$ being the angle in degrees that the frame rotating at applied field frequency, $\Omega_{rf}$ (or “freq”) moves during the time, $t_p$ (or “time”) it is applied.

$$\theta_{rf} = \Omega_{rf} t_p$$

The effective Hamiltonian (static in the rotating frame of the rf-field) is given by

$$\boldsymbol{H}_{eff} = \boldsymbol{H}_0 + \Omega_{rf}\boldsymbol{F}_{\{i\},z} - \left[\frac{\theta_p}{(360 \times t_p)}\right]\boldsymbol{F}_{\{i\},y}$$

Where $\boldsymbol{H}_0$ (or “H”) is the currently active component of the total Hamiltonian without the field, $\theta_p$ (or “theta”) is the pulse angle reguardless of the pulse length, and the two angular momentum operators, $\boldsymbol{F}_{\{i\},z}$ and $\boldsymbol{F}_{\{i\},y}$ , are obtained from summing over the associated single spin operators for all spins in the spin system “sys” of isotope type “iso”. The same spin selectivity is used to compute the z-rotation operator $\boldsymbol{R}_{\{i\},z}$ .

**See Also:**

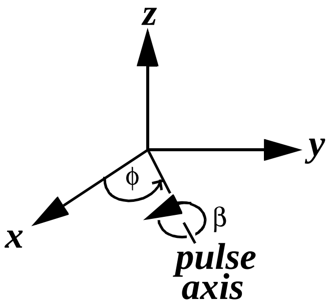
## 11.3.3 Shxypuls

**Usage:**

```
#include <gamma.h >
gen_op Shxypuls (spin_system &sys, gen_op sigma, gen_op &H,
                        char* iso, double offset, double time, double
beta, double phi)
gen_op Shxypuls_U (spin_system &sys, gen_op &H,
                        char* iso, double offset, double time, double
beta, double phi)
gen_op Shxypuls (spin_system &sys, gen_op sigma, gen_op &H, char *iso1,
double offset1,
                        char* iso2, double offset2, double time, double
beta, double phi)
gen_op Shxypuls_U (spin_system &sys, gen_op &H, char *iso1, double
offset1,
                        char* iso2, double offset2, double time, double
beta, double phi)
```

**Description:**

Applies a “soft pulse” to the density matrix of angle beta about an axis in the xy-plane phi degrees over from the x-axis affecting the spins of the isotope type(s) specified.



1. Shxypuls(spin_system &sys, gen_op &sigma, gen_op &H, char* iso, double offset, double time, double beta, double phi) - Returns the density matrix operator, sigma, after having been “pulsed” with a rotation of angle beta about the axis phi degrees over from the x-axis. The pulse is applied at a frequency “offset” from the isotope “iso” carrier frequency length “time”. The $B_1$ strength is adjusted to produce the rotation beta for a spin on resonance.
2. Shxypuls_U(spin_system &sys, gen_op &H, char* iso, double offset, double time, double beta, double phi) - Returns the propagator for a soft pulse with a rotation of angle beta about the axis phi degrees from the x-axis. The pulse is applied at a frequency “offset” from the isotope “iso” carrier frequency length “time”. The $B_1$ strength is adjusted to produce the rotation beta for a spin on resonance
3. Shxypuls(spin_system &sys, gen_op &sigma, gen_op &H, char* iso1, double offset1, char* iso2, double offset2, double time, double beta, double phi) - Returns the density matrix operator, sigma, after having been “pulsed” with a rotation of angle beta about the axis phi degrees over from the x-axis.The pulse is simultaneously applied at a frequency “offset1” from the isotope “iso1” carrier frequency and at frequency “offset2” from the isotope “iso2” carrier frequency for a length “time”. The $B_1$ strength is adjusted to produce the rotation beta for a spin on the two resonances.
4. Shxypuls_U(spin_system &sys, gen_op &H, char* iso1, double offset1, char* iso2, double offset2, double time, double beta, double phi) - Returns the propagator for a soft pulse with a rotation of angle beta about the axis phi degrees over from the x-axis. The pulse is simultaneously applied at a frequency “offset1” from the isotope “iso1” carrier frequency and at frequency “offset2” from the isotope “iso2” carrier frequency for a length “time”. The $B_1$ strength is adjusted to produce the rotation beta for a spin on the two resonances.

**Return Value:**

The function returns a general operator.

**Example(s):**

**Mathematical Basis:**

**See Also:**

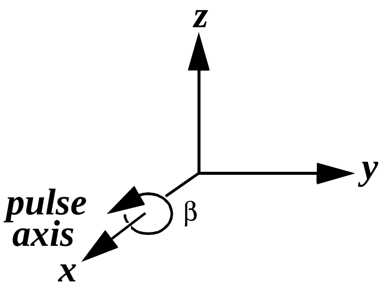
## 11.3.4 ShxpulsB

**Usage:**

```
#include <gamma.h >
gen_op ShxpulsB (spin_system &sys, gen_op sigma, gen_op &H,
                                        char* iso, double offset, double
time, double gamB1)
gen_op ShxpulsB_U (spin_system &sys, gen_op &H,
                                        char* iso, double offset, double
time, double gamB1)
```

**Description:**

Applies a “soft pulse” to the density matrix of angle $\gamma B_1 t_p$ about the x-axis affecting the spins of the isotopye type specified in the argumen..



1. SxpulsB(spin_system &sys, gen_op &sigma, char* iso, double offset, double time, double gamB1) - Returns the density matrix operator, sigma, after having been “pulsed” with a rotation of angle beta about the axis. The pulse is applied at a frequency “offset” from the isotope “iso” carrier frequency length “time”.
2. SxpulsB_U(spin_system &sys, gen_op &H, char* iso, double offset, double time, double gamB1) - Returns the propagator for a soft pulse with a rotation of angle beta about the axis. The pulse is applied at a frequency “offset” from the isotope “iso” carrier frequency length “time”.

**Return Value:**

The function returns a general operator.

**Example(s):**

```
#include <gamma.h >
gen_op sigma, H, U;                      // set up operators for density matrix, Hamilt. & prop.
spin_system AMX(3);                      // set up a three spin system.
AMX.isotope(0, “1H”);                    // set the first spin to a proton.
AMX.isotope(1, “19F”);                   // set the second spin to a fluroine.
AMX.isotope(2, “19F”);                   // set the third spin to a fluorine.
sigma = sigma_eq(AMX);                   // set the density matrix to equilibrium for AMX
H = Ho(AMX);                             // set the Hamiltonian to isotropic liquid w/ weak J het.
sigma = Sypuls(AMX, sigma, H, “19F”, 100.0, 10.0e-6, 90.0); // 90 Pulse on fluorine for 10 usec., 100 Hz.
U = Sxpuls(AMX, H, “19F”, 100.0);        // Propagator for previous pulse. Used defaults.
sigma = Sypuls(AMX, sigma, H, “19F”, 100.0, “1H”, 10.0, 10.0e-6, 90.0); // ., 100 Hz.
```

U = Sxpuls(AMX, H, “19F”, 100.0, “1H”, 10.0); // Propagator for previous pulse. Used defaults.

**Mathematical Basis:**

When the function SxpulsB returns the propagated density matrix, the fomula used is

$$\sigma^{SP} = \boldsymbol{U}^{SP}\sigma_0[\boldsymbol{U}^{SP}]^{-1}$$

where the propagator is that which is returned by the function SxpulsB_U. In turn, this propagator is given by

$$\boldsymbol{U}^{SP} = \boldsymbol{R}_{\{i\},z}(\theta_{rf})[e^{2\pi i H_{eff} t}]$$

with $\theta_{rf}$ being the angle in degrees that the frame rotating at applied field frequency, $\Omega_{rf}$ (or “freq”) moves during the time, $t_p$ (or “time”) it is applied.

$$\theta_{rf} = \Omega_{rf} t_p$$

The effective Hamiltonian (static in the rotating frame of the rf-field) is given by

$$\boldsymbol{H}_{eff} = \boldsymbol{H}_0 + \Omega_{rf}\boldsymbol{F}_{\{i\},z} - \gamma B_1 \boldsymbol{F}_{\{i\},x}$$

Where $\boldsymbol{H}_0$ (or “H”) is the currently active component of the total Hamiltonian without the field, $\theta_p$ (or “theta”) is the pulse angle reguardless of the pulse length, and the two angular momentum operators, $\boldsymbol{F}_{\{i\},z}$ and $\boldsymbol{F}_{\{i\},x}$ , are obtained from summing over the associated single spin operators for all spins in the spin system “sys” of isotope type “iso”. The same spin selectivity is used to compute the z-rotation operator $\boldsymbol{R}_{\{i\},z}$ .

**See Also: Sxpul, SypulB, SxypulB, Ixpuls, Ixpuls_U**

## 11.4 Description

If one considers an rf-field whose amplitude and phase is no longer constant (both were constant in the treatment of soft pulses) thenThe solution to the Liouville equation with an applied rf-field can be simplified under the approximation that the field is applied essentially instantaneous.

## 11.5 Chapter Source Codes

# 12 Evolution

## 12.1 Overview

The density operator $\sigma$ changes in time according to the Liouville equation. Under conditions of a static Hamiltonian its evolution is quite simplistic. The routines in this section evolve the density operator in time under static Hamiltonians. Note that many Hamiltonians may be considered "static" over a short time period, and thus time dependent problems can be treated piecewise with these functions as well. The latter is equivalent to an approximate integration of the Liouville equation.

## 12.2 Evolution Functions

**General Functions**

evclve - High-resolution isotropic NMR Hamiltonian

## 12.3 Routines

### 12.3.1 evolve

**Usage:**

```
#include <gamma.h >
void evolve(gen_op sigma, gen_op H, double time)
void evolve(gen_op sigma, gen_op U)
void evolve(gen_op sigma, super_op LOp, double time)
void evolve(gen_op sigma, super_op LOp)
void evolve_ip (gen_op sigma, gen_op H, double time)
void evolve_ip (gen_op sigma, gen_op U)
void evolve_ip (gen_op sigma, super_op L)
```

**Description:**

Evolves the operator sigma under the operator H for the time specified. Alternatively, the operator may be evolved by the propagator U. A third equivalent proceedure involves propagation by the superoperator L. These three evolutions proceed according to the following equation.

$$\sigma(t_0+t) = e^{-2\pi i\boldsymbol{H}t}\sigma(t_0)e^{2\pi i\boldsymbol{H}t} = \boldsymbol{U}\sigma(t_0)\boldsymbol{U}^{-1} = \hat{\boldsymbol{L}}\sigma(t_0) \tag{9-21}$$

Another common evolution is under a relaxation superoperator or Liouville superoperator. In this case the following evolution is provided.

$$\sigma(t_0 + t) = e^{-\hat{\boldsymbol{L}}t}\sigma(t_0) \qquad (9\text{-}22)$$

1. evolve( gen_op &sigma, gen_op H, double time) - evolves the density matrix sigma under the time independent Hamiltonian H for a time time. The evolved sigma is returned, the original unchanged.
2. evolve( gen_op &sigma, gen_op U) - evolves the density matrix sigma under the time independent propagator U. The evolved sigma is returned, the original unchanged.
3. evolve( gen_op &sigma, super_op L) - evolves the density matrix sigma under the time independent super operator propagator L. The evolved sigma is returned, the original unchanged.
4. evolve( gen_op &sigma, gen_op H, double time) - same as 1. but the input density matrix sigma is altered.
5. evolve( gen_op &sigma, gen_op U) - same as 2. but the input density matrix sigma is altered.
6. evolve( gen_op &sigma, super_op L) - same as 3. but the input density matrix sigma is altered.

## Return Value:

Either a geneal operator is returned or the function returns void (for the “in place” functions).

## Example(s):

```
#include <gamma.h >
spin_system AMX;                        // set up a spin system.
AMX.read(“AMX.sys”);                    // read spin system in from disk file AMX.sys.
gen_op sigma = sigma_eq(AMX);           // set up a density matrix to equilibrium for AMX
sigma = Ipulse_x(AMX, sigma, 90.0);     // pulse all spins by 90 degrees.
gen_op H,U;                             // set up two general operators.
H = Ho(AMX);                            // set H to isotropic liquid Hamiltonian.
U = prop(H,5.0);                        // set U to propatator for evolution under H, 5 sec.
evolve_ip(sigma, H, 5.0);               // evolve the density matrix for 5 seconds under H.
sigma = evolve(sigma, U);               // equivalent evolution to previous step.
evolve_ip(sigma, U);                    // equivalent evolution to previous step.
super_op L;                             // set up a superoperator.
L = U_transform(U);                     // set L to propagator U.
evolve_ip(sigma, U);                    // another equivalent time evolution step.
```

## See Also:

## Mathematical Basis:

The density matrix evolves in time according to the Liouville equation (von Neumann)

$$ih\frac{d\sigma}{dt} = [\boldsymbol{H},\sigma] \quad , \qquad (9\text{-}23)$$

where $\sigma$ is the density matrix and $\boldsymbol{H}$ the active Hamiltonian. When $\boldsymbol{H}$ is time independent, the solution to (9-23) is

$$\sigma(t + t_0) = e^{(-i\boldsymbol{H}t)/h}\sigma(t_0)e^{(i\boldsymbol{H}t)/h} \qquad (9\text{-}24)$$

where $t$ is the evolution time and $t_0$ the initial time. When the Hamiltonian is input in frequency units this becomes

$$\sigma(t+t_0) = e^{-i\boldsymbol{H}t}\sigma(t_0)e^{i\boldsymbol{H}t} = \boldsymbol{U}\sigma(t_0)\boldsymbol{U}^{-1} \quad . \tag{9-25}$$

Here, $\boldsymbol{U}$ is the propagator. For individual elements of $\sigma(t+t_0)$, from (9-25),

$$\langle i|\sigma(t+t_0)|j\rangle = \sum_k^{dim}\sum_l^{dim}\langle i|\boldsymbol{U}|k\rangle\langle k|\sigma(t_0)|l\rangle\langle l|\boldsymbol{U}^{-1}|j\rangle$$

This calculation is most efficiently done in the eigenbasis of $\boldsymbol{H}$, where $\boldsymbol{H}$ is represented by a diagonal matrix. In that basis, the propagator $\boldsymbol{U}$ is represented by a diagonal matrix as well, and

$$\langle i|\sigma(t+t_0)|j\rangle = \langle i|\boldsymbol{U}|i\rangle\langle i|\sigma(t_0)|j\rangle\langle j|\boldsymbol{U}^{-1}|j\rangle = \lambda_i\lambda_j^*\langle i|\sigma(t_0)|j\rangle$$

where $\lambda_i$ are the eigenvalues of $\boldsymbol{U}$ which in turn are the exponentials of the eigenvalues of $\boldsymbol{H}$.

$$\lambda_i^{\boldsymbol{U}} = \langle i|\boldsymbol{U}|i\rangle = \langle i|e^{-i\boldsymbol{H}t}|i\rangle = e^{-i\lambda_i^{\boldsymbol{H}}t}$$

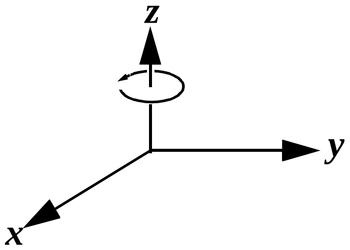
If we consider a homonuclear spin system with the active Hamiltonian $\boldsymbol{H}$ is dominated by the chemical shift term, to a good approximation

$$\boldsymbol{H} \approx \Omega\boldsymbol{F}_z$$

where $\Omega$ is the chemical shift. Equation (9-25) then becomes

$$\sigma(t+t_0) \approx e^{-i\boldsymbol{F}_z\Omega t}\sigma(t_0)e^{i\boldsymbol{F}_z\Omega t} = \boldsymbol{R}_z(\Omega t)\sigma(t_0)[\boldsymbol{R}_z(\Omega t)]^{-1} \quad . \tag{9-26}$$

Evidently, the effect of the isotropic chemical shift is to cause rotation about the z-axis.



The equivalent solution to equation (9-25), i.e. the solution to the von Neumann equation under a time independent Hamiltonian, can be written with superoperator formalism as[1]

$$\sigma(t+t_0) = e^{-i\boldsymbol{H}t}\sigma(t_0)e^{i\boldsymbol{H}t} = \boldsymbol{U}\sigma(t_0)\boldsymbol{U}^{-1} = \hat{\boldsymbol{U}}\sigma(t_0) \qquad . \tag{9-27}$$

In this context, the superoperator equivalent of the propagator $\boldsymbol{U}$, namely $\hat{\boldsymbol{U}}$, is determined from[2]

$$\hat{\boldsymbol{U}} = \boldsymbol{U} \otimes \boldsymbol{U}^* \tag{9-28}$$

Where $\boldsymbol{U}^*$ is the complex conjugate of the propagator $\boldsymbol{U}$ and $\otimes$ is a tensor product.

[1] See EBW, page 16, equation (2.1.41)
[2] See EBW, page 24, equation (2.1.83).

## 12.4 Description

The density operator equation of motion (Liouville, von Neumann) is

$$ih\frac{d\sigma}{dt} = [\boldsymbol{H},\sigma] \tag{9-29}$$

where $\boldsymbol{H}$ is the acting Hamiltonian and $\sigma$ the denisty operator for some evolving system. It is easy to verify that the solution to this equation, under a time independent Hamiltonian, is

$$\sigma(t) = e^{-i\boldsymbol{H}t}\sigma_o e^{i\boldsymbol{H}t}. \tag{9-30}$$

Here t is the time during which the Hamiltonian has acted on the spin system, $\sigma_o$ represents the initial state of the system and $\sigma(t)$ the final state. It is common to write this in terms of the propagator $\boldsymbol{U}$,

$$\sigma(t) = \boldsymbol{U}\sigma_o\boldsymbol{U}^{-1} \qquad \text{where} \qquad \boldsymbol{U} = e^{-i\boldsymbol{H}t} \tag{9-31}$$

The Liouville equation may also be written using a commutation superoperator

$$ih\frac{d}{dt}|\sigma\rangle = \hat{\boldsymbol{H}}|\sigma\rangle \tag{9-32}$$

where the density operator acts as a superket in Liouville space. In this case the commutation superopertor is given by

$$\hat{\boldsymbol{H}} = \boldsymbol{H}\otimes\boldsymbol{H}^{*} \tag{9-33}$$