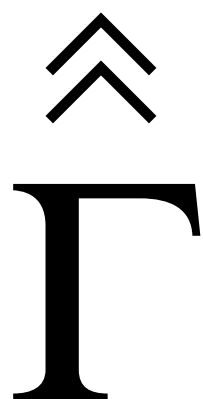


GAMMA

Liouville Space Module



Author: Dr. Scott A. Smith, Tilo Levante
Date: March 16, 2000

Table of Contents

1	<i>Class Super Operator (super_op)</i>	4
1.1	Overview	4
1.2	Available Superoperator Functions	4
1.3	Arithmetic Operators	6
1.4	Complex Functions	14
1.5	Basis Manipulations	20
1.6	Description	22
2	<i>Dynamic Spin Systems</i>	29
2.1	Overview	29
2.2	Available Dynamic Spin System Functions	29
2.3	Correlation Time Functions	31
2.4	Dipole Functions	33
2.5	Quadrupole Functions	37
2.6	Description	38
2.7	Dynamic System Parameter Files	39

1 Introduction

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

1 Class Super Operator (super_op)

1.1 Overview

The *Class SUPER OPERATOR* defines all the necessary attributes of a quantum mechanical superoperator, **LOp**. Within *Class SUPER OPERATOR* are also specifications of superoperator properties (dimension, ...), algebras (+, *,...), and definitions of all available superoperator functions.

1.2 Available Superoperator Functions

Superoperator Basic Functions

super_op	- Constructor:	LOp, LOp(mx), LOp(mx1, mx2), LOp(mx, bs), LOp(LOp1).	page 6
+	- Addition:	LOp + LOp, LOp + mx, mx + LOp.	page 8
+=	- Unary Addition:	+= LOp, += mx.	page 8
-	- Subtraction:	LOp - LOp, LOp - mx, mx - LOp.	page 9
-=	- Unary Subtraction:	-= LOp, -= mx.	page 10
*	- Multiplication:	LOp * LOp, LOp*mx, mx*LOp, LOp*Op,z*LOp,LOp*z.	page 10
*=	- Unary Multiplication:	*=LOp, *=mx, *=z.	page 11
-	- Negation:	-LOp.	page 9
=	- Assignment:	= LOp, = mx.	page 7
/	- Division:	LOp / z.	page 12
/=	- Unary Division:	/= z.	

Complex Superoperator Functions¹

left	- Left Translation LOp:	Op, mx	page 13
right	- Right translation LOp:	Op, mx	page 14
commutator	- Commutation LOp:	[Op,], [mx,]	page 15
d_commutator	- Double commutation LOp:	[Op, [Op,]], [mx, [mx,]], [Op1, [Op2,]], [mx1, [mx2,]]	page 16
U_transform	- Unitary transform LOp:	$U_U^{-1} : LOp*Op = U*Op*U^{-1}$	page 17
exp	- Superoperator exponential:	exp(LOp);	page 18

Superoperator Internal Access

get_mx	- Retrieve superoperator matrix:	LOp
put_mx	- Input superoperator matrix:	LOp
put_basis	- Input superoperator basis:	LOp
()	- Direct LOp element access:	z = LOp(3,2).

1. The commutation and double commutations superoperators may take spin operators instead of operators or matrices. For the unitary transformation superoperator U is either an operator, spin operator, or matrix.

put - Input LOp element directly:
get - Direct LOp element access:

Basis Manipulations

set_EBR	- Put LOp into its eigenbasis: LOp	page 19
set_HBR	- Put LOp into its default Liouville space basis.	page 19
Op_base	- Put LOp into the Liouville space basis of another superoperator. - Put operator into the Hilbert space basis of a superoperator.	

Superoperator I/O

<<	- Send LOp to an output stream: LOp	page 19
----	--	---------

1.3 Arithmetic Operators

1.3.1 super_op

Usage:

```
#include <super_op.h>
super_op ( )
super_op (matrix &mx)
super_op (matrix &mx1, matrix &mx2)
super_op (matrix &mx, basis &bs)
super_op (super_op &LOp)
```

Description:

The function *super_op* is used to create a superoperator quantity.

1. *super_op()* - sets up an empty superoperator which can later be explicitly specified.
2. *super_op(mx)* - sets up a superoperator with the matrix in the argument assumed to be the superoperator in the default basis. The specified matrix must be a square array in the Liouville space.
3. *super_op(mx1, mx2)* - With two matrices as arguments, the function sets up a superoperator with matrix representation **mx1** in the basis formed from matrix **mx2**. Again, **mx1** must be a square matrix of the Liouville space dimension. The basis should relate properly to the default basis, and be the Hilbert space dimension.
4. *super_op(mx, bs)* - With a matrix (**mx**) and a basis (**bs**) as arguments, the function sets up a superoperator with matrix representation **mx** in the basis **bs**. Again, **mx** must be a square matrix and its dimension that of the Liouville space. The basis should relate properly to the default basis, and be the Hilbert space dimension.
5. *super_op(LOp)* - One may produce a superoperator from another superoperator. The new superoperator will then be equivalent to the input superoperator **LOp**.

Return Value:

Creates a new superoperator which may be subsequently used with all defined superoperator functions.

Examples:

```
#include <super_op.h>
matrix mx;
basis bs;
super_op LOp;                // produces an empty superoperator LOp.
super_op LOp1(mx);           // set superoperator LOp, matrix mx in default basis.
super_op LOp2(mx, bs);       // set superoperator LOp2, matrix mx in the basis bs.
super_op LOp3(LOp2);         // set superoperator LOp3 equal to current LOp2.
```

Mathematical Basis:

Each superoperator is a matrix in Liouville space and has associated with it a basis in Hilbert space.

In all superoperator constructors the matrix provided must be in the Liouville space dimension, the Hilbert space dimension squared. The constructor basis (or matrix to be used as the basis) must be of Hilbert space dimension, smaller in size than the actual superoperator matrix.

Superoperator Matrix in Liouville Space

11	12	13	1N ²
21	22	23	2N ²
31	32	33	3N ²
41	42	43	4N ²
⋮	⋮	⋮		⋮
⋮	⋮	⋮		⋮
⋮	⋮	⋮		⋮
⋮	⋮	⋮		⋮
⋮	⋮	⋮		⋮
⋮	⋮	⋮		⋮
⋮	⋮	⋮		⋮
N ² 1	N ² 2	N ² 3	N ² N ²

Superoperator Basis in Hilbert Space

11	12	13	...	1N
21	22	23	⋮	2N
31	32	33	⋮	3N
⋮	⋮	⋮		⋮
⋮	⋮	⋮		⋮
⋮	⋮	⋮		⋮
⋮	⋮	⋮		⋮
N1	N2	N3	...	NN

See Also: =

1.3.2 =

Usage:

```
#include <super_op.h>
super_op operator = (super_op& LOp)
super_op operator = (matrix& mx)
```

Description:

This allows for the ability to equate either a superoperator or a matrix to another superoperator.

1. For the equating of two superoperators, **LOp** = **LOp1**, superoperator **LOp** is set equal to superoperator **LOp1**. **LOp** will be in the current basis of **LOp1**.
2. For the assignment of a matrix to a superoperator, **LOp** = **mx**, the superoperator **LOp** is set equal to the matrix **mx** and the basis is assumed to be the default basis. The matrix **mx** must be in the Liouville space.

Return Value:

none.

Example(s):

```
#include <super_op.h>
matrix mx;                                // define a matrix mx.
super_op LOp1, LOp2;                      // define two superoperators LOp1 and LOp2.
LOp1 = LOp2;                              // LOp1 set equal to superoperator LOp2.
LOp1 = mx;                                // LOp1 set in the default basis to have matrix mx.
```

See Also: **super_op** (constructors)

1.3.3 +

Usage:

```
#include <super_op.h>
super_op operator + (super_op& LOp1, super_op& LOp2)
super_op operator + (super_op& LOp, matrix& mx)
super_op operator + (matrix& mx, super_op& LOp)
```

Description:

This allows for the addition of two superoperators, **LOp1 + LOp2** and the addition of a superoperator with a matrix, **LOp1 + mx**.

1. **LOp1 + LOp2** - Definition of the addition of two superoperators **LOp1** and **LOp2**. A check is made to insure that both superoperators are in the same basis. If this is not true, superoperator **LOp2** is transformed into the basis of superoperator **LOp1** prior to the addition, thus insuring that the addition *produces a result in (and only in) the same basis of LOp1*.
2. **LOp + mx** - Definition of the addition of a matrix **mx** to a superoperator **LOp**. The matrix **mx** is assumed to be a matrix in the default basis and the addition takes place in the default basis. Superoperator **LOp** is first placed in the default basis, the addition takes place, and then the result is *new superoperator in the default basis*.
3. **mx + LOp** - Definition of the addition of a superoperator **LOp** to a matrix **mx**. The result is equivalent to the previous addition, it produces a *new superoperator in the default basis*.

Return Value:

A new superoperator which exists in an appropriate representation.

Example(s):

```
#include <super_op.h>
matrix mx;
super_op LOp1,LOp2,LOp3;           // define three superoperators LOp1,LOp2, & LOp3.
LOp3 = LOp1 + LOp2;                 // LOp3 is sum LOp1 + LOp2. Only in WB of LOp1.
LOp3 = mx + LOp1;                   // LOp3 is sum LOp1 + matrix mx. Only in DB.
LOp3 = LOp1 + mx;                   // same as the previous line.
```

See Also:

+=, -, -=

1.3.4 +=

Usage:

```
#include <super_op.h>
super_op operator += (super_op &LOp)
super_op operator += (matrix &mx)
```

Description:

This allows for the addition of two superoperators of the type **LOp = LOp + LOp1** or the addition of a matrix

to a superoperator **LOp** = **LOp** + **mx**.

1. **LOp** += **LOp1** - Definition of the unary addition of two superoperators **LOp** and **LOp1**. A check is made to insure that both superoperators are in the same basis. If this is not true, **LOp1** is placed into the basis of **LOp** prior to the addition, thus insuring that the subtraction *produces a result in (and only in) the same basis of LOp*.
2. **LOp** += **mx** - Unary addition of a matrix **mx** to a superoperator **LOp**. The matrix **mx** is assumed to be a superoperator matrix in the default basis and the addition takes place in the default basis. Superoperator **LOp** is first placed in the default basis, the matrix **mx** is then added to it.

Use of this operator is more computationally efficient than the two step operation. That is, the statement **LOp** += **LOp1**; is preferred over the statement **LOp** = **LOp** + **LOp1**;

Return Value:

A new superoperator which exists in an appropriate representation.

Example(s):

```
#include <super_op.h>
matrix mx;
super_op LOp, LOp1;           // define two superoperators LOp and LOp1.
LOp += LOp1;                  // LOp1 added to LOp. Only in the WB of LOp.
```

See Also: +, -, -=

1.3.5 -

Usage:

```
#include <super_op.h>
super_op LOp - (super_op& LOp, super_op& LOp1)
super_op LOp - (super_op& LOp, matrix& mx)
super_op LOp - (matrix& mx, super_op& LOp)
super_op LOp - (super_op& LOp)
```

Description:

This allows for the subtraction of two superoperators **LOp** and **LOp1**, for the subtraction of a matrix from a superoperator, and for the negation of a superoperator.

1. **LOp** - **LOp1** - Definition of the subtraction of two superoperators **LOp** and **LOp1**. A check is made to insure that both superoperators are in the same basis. If this is not true, **LOp1** is placed into the basis of **LOp** prior to the subtraction, thus insuring that the subtraction *produces a result in (and only in) the same basis of LOp*.
2. **LOp** - **mx** - Definition of the subtraction of a matrix **mx** from a superoperator **LOp**. The matrix **mx** is assumed to be a matrix in the default basis and the subtraction takes place in the default basis. Superoperator **LOp** is first placed in the default basis, the matrix **mx** is then subtracted from it, and then the result is *new superoperator in the default basis*.
3. **mx** - **LOp** - Definition of the subtraction of a superoperator **LOp** from a matrix **mx**. The result is equivalent to the negative of the previous subtraction, it produces a *new superoperator in the default basis*.
4. - **LOp** - Definition of the negation of superoperator **LOp**. The result is a superoperator, in (and only in) the working basis of **LOp**, which is -1.0 * **LOp**.

Return Value:

A new superoperator which exists in an appropriate representation.

Example(s):

```
#include <super_op.h>
matrix mx;
super_op LOp, LOp1, LOp2;           // define three superoperators LOp, LOp1, and LOp2.
LOp2 = LOp - LOp1;                  // LOp2 set to LOp - LOp1. Only in the WB of LOp.
LOp2 = LOp - mx;                    // LOp2 set to be LOp minus matrix mx. Only in DB
LOp2 = mx - LOp;                    // same as -(LOp - mx). Only in the DB of LOp.
LOp2 = -LOp;                        // LOp2 set to negative LOp. Only in the WB of LOp.
```

See Also: +, +=, -=s

1.3.6 -=

Usage:

```
#include <super_op.h>
void operator -= (super_op& LOp1)
```

Description:

This allows for the subtraction of two superoperators of the type **LOp = LOp - LOp1**. A check is made to insure that both superoperators are in the same basis. If this is not true, **LOp1** is transformed into the basis of **LOp** prior to the addition, thus insuring that the subtraction *produces a result in (and only in) the same basis of LOp*. Use of this operation is more computationally efficient than the two step operation. That is, the statement **LOp -= LOp1**; is preferred over the statement **LOp = LOp - LOp1**;

Return Value:

Void. The input superoperator is modified.

Example(s):

```
#include <super_op.h>
matrix mx;
super_op LOp, LOp1;                 // define two superoperators LOp and LOp1.
LOp -= LOp1;                        // LOP set to LOp - LOp1. Only in WB of LOp.
```

See Also:

-, +, +=

1.3.7 *

Usage:

```
#include <super_op.h>
super_op operator * (super_op& LOp1, super_op& LOp2)
super_op operator * (super_op& LOp, matrix& mx)
super_op operator * (matrix& mx, super_op& LOp)
super_op operator * (complex& z, super_op& LOp)
super_op operator * (super_op& LOp, complex& z)
gen_op operator * (super_op& LOp, gen_op &Op)
```

Description:

This allows for the multiplication of two superoperators **LOp1** and **LOp2**, the multiplication of a superoperator and a matrix, and for the multiplication of a scalar and a superoperator. Additionally the multiplication of a superoperator into a general operator is defined, but not the reverse.

1. For the multiplication of two superoperators, a check is made to insure that both superoperators are in the same basis. If this is not true, **LOp2** is transformed into the basis of **LOp1** prior to the multiplication, thus insuring that the multiplication *produces a result in the same basis of LOp1*. Here, the order of the superoperators can make a difference in the result.
2. For the multiplication of a superoperator times a matrix, a superoperator is produced. It is assumed that the matrix is a quantity in the default basis so that **LOp** is changed into the default basis before the multiplication.
3. The multiplication of a matrix times a superoperator also produces a superoperator. The treatment is similar to the previous usage except the ordering can make a difference.
4. For the multiplication of a scalar times a superoperator, the complex scalar *z* is multiplied into each element of **LOp** to produce a superoperator in the same basis of **LOp**.
5. The multiplication of a superoperator times a scalar produces the same result as multiplication of a scalar times a superoperator.
6. The multiplication of a superoperator into a general operator produces a new operator. This operation is performed in the basis of the superoperator.

Return Value:

none.

Example(s):

```
#include <super_op.h>
complex z;
super_op LOp1,LOp2,LOp3;           // define three superperators LOp1, LOp2, and LOp3.
LOp3 = LOp1 * LOp2;                // C is product of LOp1 times LOp2. In WB of LOp1.
LOp3 = LOp2 * LOp1;                // maybe not same as the previous line, order matters.
LOp3 = LOp1 * z;                   // LOp3 is LOp1 with all elements multiplied by z.
LOp3 = z * LOp1;                   // same result as previous line. Only in WB of LOp1.
gen_op Op1;                         // define two general operators Op1 and Op2.
Op1 = LOp1 * Op2;                   // Op1 is LOp1 multiplied into Op2. In WB of LOp1.
```

See Also: *=, +, -, /

1.3.8 *=**Usage:**

```
#include <super_op.h>
super_op operator *= (super_op& LOp1)
super_op operator *= (matrix& mx)
super_op operator *= (complex& z)
```

Description:

This allows for the multiplication of two superoperators of the type **LOp1 = LOp1 * LOp2** or of a matrix with a superoperator, or of a scalar with a superoperator.

1. **LOp2 *= LOp1** - For the multiplication of two superperators, a check is made to insure that both Oper-

ators are in the same basis. If this is not true, **LOp1** is transformed into the basis of **LOp2** prior to the multiplication, thus insuring that the multiplication *produces a result in the same basis of LOp2*. Here, the order of the superoperators can make a difference in the result.

2. **LOp *= mx** - For the multiplication of a superoperator times a matrix, an superoperator is produced. It is assumed that the matrix is a quantity in the default basis so that **LOp** is changed into the default basis before the multiplication.
3. **LOp *= z** - The multiplication of a superoperator times a scalar (complex) also produces a superoperator. This operation *produces a result exclusively in the original working basis of LOp*.

Use of this operation is more computationally efficient than the two step operation. That is, the statement **LOp1 *= LOp2**; is preferred over the statement **LOp1 = LOp1 * LOp2**;

Return Value:

A new superoperator which exists in an appropriate representation.

Example(s):

```
#include <super_op.h>
matrix mx;
super_op LOp1,LOp2;           // define two superoperators LOp1 and LOp2.
LOp1 *= LOp2;                 // LOp1 set to LOp1 * LOp2. Only in WB of LOp1.
```

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

1.3.9 /

Usage:

```
#include <super_op.h>
super_op operator / (super_op& LOp, complex& z)
```

Description:

This allows for the division of a superoperator by a complex number. Each element of the superoperator matrix is divided by the complex number. The operation *produces a result exclusively in the original working basis of LOp*.

Return Value:

A new superoperator which exists in an appropriate representation.

Example(s):

```
#include <super_op.h>
complex z;
super_op LOp;           // define a superoperator LOp.
LOp *= z;                // LOp1 set to LOp * z. Only in WB of LOp.
```

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

1.4 Complex Functions

1.4.1 left

Usage:

```
#include <super_op.h>
super_op left (gen_op &Op)
```

Description:

Computes the left translation superoperator as defined by equation (14-5),

$$\Gamma A = OpA.$$

The output superoperator is in the working basis of the input operator.

Return Value:

The function returns a superoperator which is the left translation superoperator.

Example(s):

```
#include <super_op.h>
int main ()
{
    spin_sys sys(1);
    gen_op Op,Op1;           // construct two null operators
    super_op LOp;            // construct a null superoperator.
    Op = Fz(sys).             // set Op to the operator Fz for the spin system.
    Op1 = Fx(sys).            // set Op1 to the operator Fx for the spin system.
    LOp = left(Op);           // LOp is the left translation superoperator for Fz.
    cout << LOp*Op1;          // should output matrix Fz*Fx.
    cout << Op*Op1;           // should also aout Fz*Fx.
}
```

This example program prints out the following Hilbert space matrix twice, Fz*Fx.

$$\frac{1}{4} \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$$

Mathematical Basis:

The left translation superoperator is computed from taking cross products of the operator **Op** with the identity matrix in accordance with equation (14-6)

$$\Gamma = Op \otimes E$$

1.4.2 right

Usage:

```
#include <super_op.h>
super_op right (gen_op &Op)
```

Description:

Computes the left translation superoperator as defined by equation (14-5),

$$\Gamma A = AOp.$$

The output superoperator is in the working basis of the input operator.

Return Value:

The function returns a superoperator which is the right translation superoperator.

Example(s):

```
#include <super_op.h>
int main ()
{
    spin_sys sys(1);
    gen_op Op,Op1;           // construct two null operators
    super_op LOp;            // construct a null superoperator.
    Op = Fz(sys).             // set Op to the operator Fz for the spin system.
    Op1 = Fx(sys).           // set Op1 to the operator Fx for the spin system.
    LOp = right(Op);         // LOp is the right translation superoperator for Fz.
    cout << LOp*Op1;         // should output matrix Fx*Fz.
    cout << Op1*Op;          // should also aout Fx*Fz.
}
```

This example program prints out the following Hilbert space matrix twice, Fx*Fz.

$$\frac{1}{4} \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}$$

Mathematical Basis:

The right translation superoperator is computed from taking a cross products of the operator Op with the identity matrix in accordance with equation (14-6),

$$\Gamma = E \otimes Op^T.$$

1.4.3 commutator

Usage:

```
#include <super_op.h>
super_op commutator (gen_op &Op)
```

Description:

Computes the superoperator equivalent to the commutator of Op as defined by equation (14-3),

$$\Gamma A = [Op, A] = OpA - AOp.$$

The output superoperator is in the working basis of the input operator.

Return Value:

The function returns a superoperator which is the commutation superoperator.

Example(s):

```
#include <super_op.h>
int main ()
{
    spin_sys sys(1);
    gen_op Op;                // construct a null operator
    super_op LOp;             // construct a null superoperator.
    Op = Fx(sys).              // set Op to the operator Fx for the spin system.
    LOp = commutator(Op);      // LOp is the commutation superoperator for [Fx, ].
    cout << LOp;              // output commutation superoperator.
}
```

This example program prints out the commutation superoperator for Fx^1 .

$$\begin{bmatrix} 0 & -0.5 & 0.5 & 0 \\ -0.5 & 0 & 0 & 0.5 \\ 0.5 & 0 & 0 & -0.5 \\ 0 & 0.5 & -0.5 & 0 \end{bmatrix}$$

Mathematical Basis:

The commutation superoperator is computed from taking cross products of the operator Op with the identity matrix in accordance with equation (14-4)

$$\Gamma = (Op \otimes E) - (E \otimes Op^T)$$

1. This matrix is listed in Ernst, Bodenhausen, and Wokaun, page 24, Figure 2.1.2

For a single spin 1/2 particle in the product basis, three commutation superoperators are

$$\hat{I}_x = \frac{1}{2} \begin{bmatrix} 0 & -1 & 1 & 0 \\ -1 & 0 & 0 & 1 \\ 1 & 0 & 0 & -1 \\ 0 & 1 & -1 & 0 \end{bmatrix} \quad \hat{I}_y = \frac{i}{2} \begin{bmatrix} 0 & -1 & -1 & 0 \\ 1 & 0 & 0 & -1 \\ 1 & 0 & 0 & -1 \\ 0 & 1 & 1 & 0 \end{bmatrix} \quad \hat{I}_z = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

According to the literature, if the commutator is Hermitian then the commutator superoperator will be Hermitian as well¹

1.4.4 d_commutator

Usage:

```
#include <super_op.h>
super_op d_commutator (gen_op &Op)
super_op d_commutator (gen_op &Op1, gen_op &Op2)
```

Description:

Computes the superoperator equivalent to the double commutator of Op. The double commutation superoperator is defined by equation (14-7),

$$\Gamma A = [Op1, [Op2, A]] ,$$

where Γ is a superoperator and both **Op1**, **Op2**, and **A** are general operators

1. d_commutator (gen_op &Op) - The double commutation superoperator is formed in the basis of the operator Op. The operators in the commutator are the same.
2. d_commutator (gen_op &Op1, gen_op &Op2) - The double commutation superoperator is formed from the two input operators. Op2 will be the operator in the inner commutator and Op1 in the outer commutator. The returned superoperator will be in the basis of Op1.

Return Value:

The function returns a superoperator which is the double commutator superoperator.

Example(s):

```
#include <super_op.h>
int main ()
{
    spin_sys sys(1);
    gen_op Op1, Op2;           // construct two null operators.
    super_op LOp;              // construct a null superoperator.
    Op1 = Fp(sys).              // set Op1 to the operator F+ for the spin system.
    Op2 = Fm(sys).              // set Op2 to the operator F- for the spin system.
    LOp = d_commutator(Op1);    // double commutation superoperator [Op1[Op1, ].
    LOp = d_commutator(Op1,Op2); // double commutation superoperator [Op1[Op2, ].
}
```

1. Ernst, Bodenhausen, Wokaun, page 20, below equation (2.1.61)

Mathematical Basis:

The double commutation superoperator is computed from cross products of the operators involved with the identity matrix as given by equation (14-9).

$$\Gamma = \Gamma 1 \Gamma 2 = (\mathbf{Op} 1 \mathbf{Op} 2 \otimes \mathbf{E}) - (\mathbf{Op} 1 \otimes \mathbf{Op} 2^T) - (\mathbf{Op} 2 \otimes \mathbf{Op} 1^T) + (\mathbf{E} \otimes \mathbf{Op} 1^T \mathbf{Op} 2^T)$$

When both operators are equivalent the equation slightly simplifies to equation (14-10).

$$\Gamma = \Gamma 1 \Gamma 1 = (\mathbf{Op}^2 \otimes \mathbf{E}) - 2(\mathbf{Op} \otimes \mathbf{Op}^T) + (\mathbf{E} \otimes \mathbf{Op}^{T^2})$$

Examples of double commutation superoperators for a single spin 1/2 particle in the product basis are given below.

$$\begin{aligned} \hat{I}_{xx} = [I_x, [I_x, \cdot]] &= \frac{1}{2} \begin{bmatrix} 1 & 0 & 0 & -1 \\ 0 & 1 & -1 & 0 \\ 0 & -1 & 1 & 0 \\ -1 & 0 & 0 & 1 \end{bmatrix} & \hat{I}_{yy} = [I_y, [I_y, \cdot]] &= \frac{1}{2} \begin{bmatrix} 1 & 0 & 0 & -1 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ -1 & 0 & 0 & 1 \end{bmatrix} & \hat{I}_{zz} = [I_z, [I_z, \cdot]] &= \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \\ \hat{I}_{xy} = [I_x, [I_y, \cdot]] &= \frac{i}{2} \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & -1 & -1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} & \hat{I}_{yz} = [I_y, [I_z, \cdot]] &= \frac{i}{2} \begin{bmatrix} 0 & -1 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & -1 & 0 \end{bmatrix} & \hat{I}_{zx} = [I_z, [I_x, \cdot]] &= \frac{1}{2} \begin{bmatrix} 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 1 \\ -1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} \end{aligned}$$

1.4.5 U_transform

Usage:

```
#include <super_op.h>
super_op U_transform(gen_op &Op)
```

Description:

Computes the superoperator equivalent to the similarity transform (unitary transformation) as defined by equation (14-11),

$$\Gamma \mathbf{A} = \mathbf{Op} \mathbf{A} \mathbf{Op}^{-1}.$$

The output superoperator is in the working basis of the input operator.

Return Value:

The function returns a superoperator.

Example(s):

```
#include <super_op.h>
gen_op Op; // construct a null operator
super_op LOp; // construct a null superoperator.
Op = Rx(sys, 90). // Op to 90 degree x-rotation operator for sys.
LOp = U_transform(Op); // LOp unitary transform superoperator Rx __ Rx-1.
```

Mathematical Basis:

The superoperator in this case relates to the operator \mathbf{Op} according to equation (14-12),

$$\Gamma = \mathbf{Op} \otimes \mathbf{Op}^*,$$

1.4.6 exp

Usage:

```
#include <super_op.h>
super_op exp(gen_op &LOp)
```

Description:

Computes the exponential of the input superoperator.,

$$\Gamma = \exp(\Gamma_1) .$$

The output superoperator is in the basis of the input superoperator.

Return Value:

The function returns a superoperator.

Example(s):

```
#include <super_op.h>
gen_op Op;                                // construct a null operator
super_op LOp;                             // construct a null superoperator.
Op = Fx(sys).                             // Op to Fx for the spin system sys
LOp = d_commutator(Op);                   // LOp double commutator {Fx, [Fx, ]].
LOp = exp(Op);                           // LOp exponential of {Fx, [Fx, ]].
```

Mathematical Basis:

1.5 Basis Manipulations

1.5.1 set_EBR

Usage:

```
#include <super_op.h>
super_op set_EBR()
```

Description:

Places the superoperator into its eigenbasis. The superoperator will then be a diagonal array in Liouville space and be associated with a basis transformation array which will convert it back into the original basis in which the superoperator was before the function call. The previous representation is destroyed but can be regenerated with the function set_HSBP.

Return Value:

The function is void. It changes the input superoperator into its eigenbasis.

Example(s):

```
#include <super_op.h>
super_op LOp;                // construct a null superoperator.
LOp.set_EBR();               // LOp placed into its eigenbasis.
```

Mathematical Basis:

1.5.2 set_HBR

Usage:

```
#include <super_op.h>
super_op set_HBR()
```

Description:

Places the superoperator into its original Hilbert space basis. If for some reason the superoperator basis has been changed from that which it had when formulated, this function will return it to its original basis.

Return Value:

The function is void. It changes the input superoperator into its original Hilbert space basis.

Example(s):

```
#include <super_op.h>
super_op LOp;                // construct a null superoperator.
LOp.set_HBR();               // LOp placed into its eigenbasis.
```

Mathematical Basis:

1.5.3 <<

Usage:

```
#include <super_op.h>
ostream& operator << (ostream& str, super_op &LOp)
```

Description:

Sends the superoperator given in the argument list to the output stream specified.

Return Value:

None, the superoperator is put into the output stream

1.6 Description

As the name implies, *Class super_op* (*SUPER OPERATOR*) deals with superoperators. Throughout this document, use is made of the symbol **LOp** for a superoperator (**Op** for operator and **L** for Liouville space). In mathematical expressions usually the capital gamma is the symbol for denoting a superoperator, Γ .

Basic Structure

Each superoperator is a matrix (*see Class MATRIX*) in Liouville space and has associated with it a basis (*see Class BASIS*) in Hilbert space.

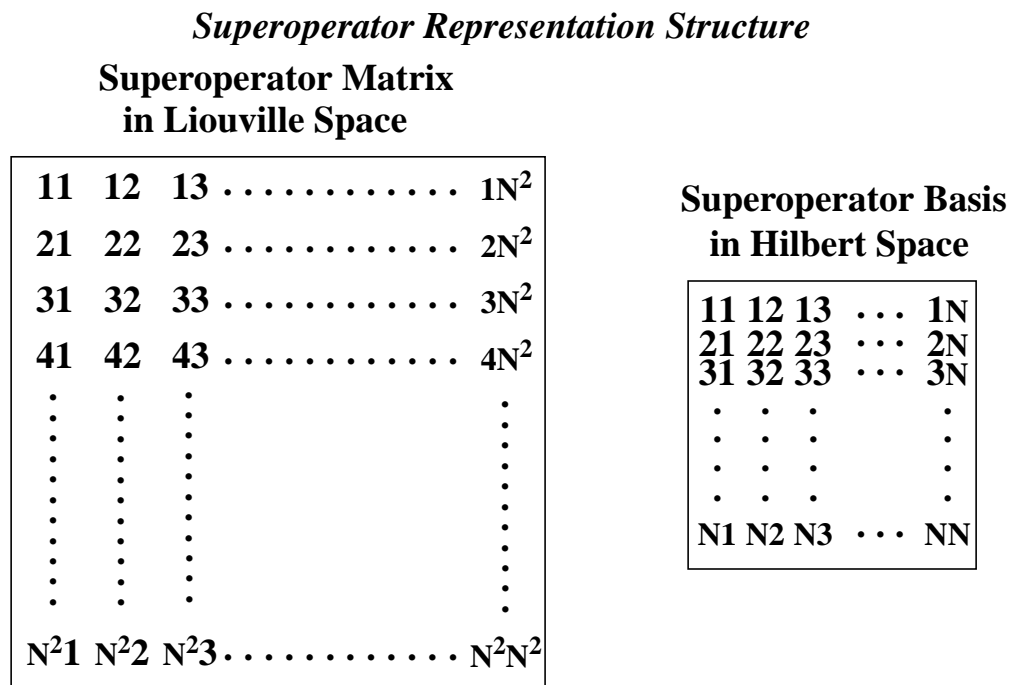


Figure 10-1 - Each superoperator has only one representation: a matrix and a basis. Any change to the representation (basis change) destroys the previous representation.

The matrix form of **LOp** depends upon which basis the superoperator is expressed in. Without knowledge of the basis the superoperator matrix is nearly useless, it could not be freely applied without careful consideration from the external user. *Class super_op* will always associate an **LOp** matrix with an **LOp** basis.

General Operators as Superoperators

There is an intrinsic relationship between general operators (spanning Hilbert space) and superoperators (spanning Liouville space). The following diagram shows the relationship between a general operator, **Op**, and a superoperator **LOp** formed (through one of the provided functions) from **Op**.

Formation of Superoperators from Operators

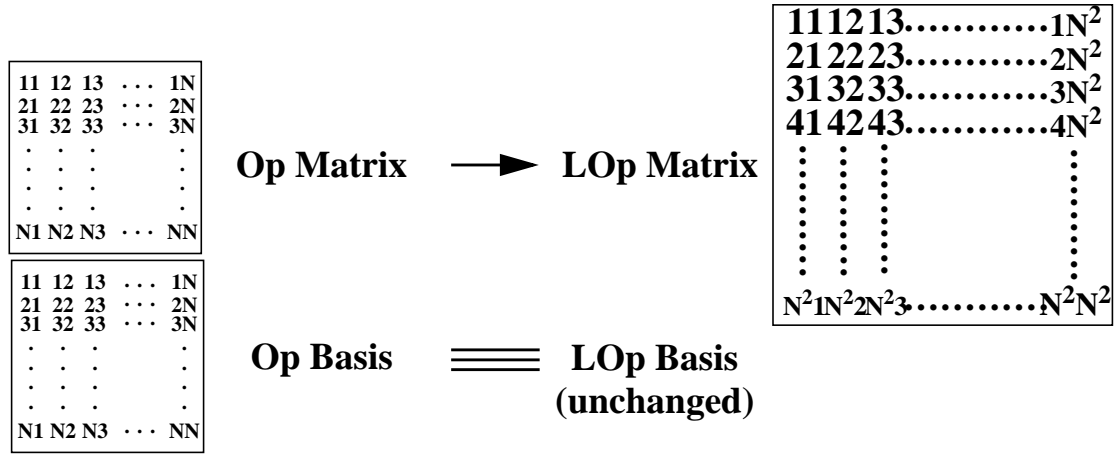


Figure 10-2 - When a superoperator is formed from an operator (or operators) its dimension is the square of the operator. On the other hand, the basis for both remain in the Hilbert space having the operator dimension.

Operator matrices are smaller in size than related superoperator matrices. Yet the basis matrices for Op and LOp can be identical and exist in Hilbert space. Since superoperators are usually formed from operators it is not a problem that the superoperator basis remains in Hilbert space. Operators will always be converted to the superoperator basis with the Hilbert space transformation matrix, *i.e.* the basis. An operator in an arbitrary basis (AB superscript) can be converted into the basis of the superoperator (SB superscript) by first transforming into the default basis (DB superscript) and then into SB.

$$(U^{AB})^\dagger \mathbf{Op}^{AB} U^{AB} = \mathbf{Op}^{DB} \quad (14-1)$$

$$U^{SB} \mathbf{Op}^{AB} (U^{SB})^\dagger = \mathbf{Op}^{SB} \quad (14-2)$$

Here **U** is a transformation matrix or basis matrix (see *Class BASIS*) in Hilbert space, **Op** the Operator matrix in Hilbert space and **U^{SB}** the basis in which the superoperator resides, also a Hilbert space array. Thus, it is generally unnecessary to change the superoperator basis. Unlike general operators, superoperators reside in only one basis at a time.

The Commutation Superoperator

The commutation superoperator is defined to be¹,

$$\Gamma \mathbf{A} = [\mathbf{Op}, \mathbf{A}] = \mathbf{Op} \mathbf{A} - \mathbf{A} \mathbf{Op} \quad (14-3)$$

where Γ is a general superoperator (not explicitly a commutation superoperator) and both **Op** and **A** are general operators. The superoperator in this case relates to the operator **Op** according to

$$\Gamma = (\mathbf{Op} \otimes \mathbf{E}) - (\mathbf{E} \otimes \mathbf{Op}^T) \quad (14-4)$$

1. Ernst, Bodenhausen, and Wokaun, page 23, equation (2.1.81)

The superscript T implies the transpose, \otimes the tensor product, and E the identity matrix of dimension equivalent to Op (the Hilbert space dimension).

The Left and Right Translation Superoperators

The commutation superoperator defined in equation (14-3) can also be written as a difference between the left and right translation superoperators¹,

$$\Gamma A = [Op, A] = OpA - AOp = \Gamma^L A - \Gamma^R A \quad (14-5)$$

where Γ^L is the left translation superoperator and Γ^R the right translation superoperator. Again Op and A are general operators. Evidently the two translation superoperators are

$$\Gamma^L = Op \otimes E \quad \text{and} \quad \Gamma^R = E \otimes Op^T \quad (14-6)$$

The superscript T implies the transpose, \otimes the tensor product, and E the identity matrix of dimension equivalent to Op (the Hilbert space dimension).

The Double Commutation Superoperator

The double commutation superoperator is defined to be,

$$\Gamma A = [Op1, [Op2, A]] \quad (14-7)$$

where Γ is a superoperator and both $Op1$, $Op2$, and A are general operators. The single commutation superoperator is also defined (see function commutator) and relates to the operator Op according to

$$\Gamma' A = [Op, A] = OpA - AOp, \quad \text{where} \quad \Gamma' = (Op \otimes E) - (E \otimes Op^T) .$$

The superscript T implies the transpose, \otimes the tensor product, and E the identity matrix of dimension equivalent to Op (the Hilbert space dimension). One can expand the double commutator in terms of single commutators and apply the single commutator superoperators twice.

$$\Gamma A = [Op1, [Op2, A]] = Op1[Op2, A] - [Op2, A]Op1$$

$$\Gamma A = Op1(\Gamma2A) - (\Gamma2A)Op1 = [Op1, (\Gamma2A)] = \Gamma1\Gamma2A$$

Thus, the double commutator could be produced by the superoperator product in the previous equation, namely

$$\Gamma = \Gamma1\Gamma2, \quad (14-8)$$

where Γ is the double commutation superoperator, $\Gamma1$ is the single commutation superoperator for the outside commutator, and $\Gamma2$ is the single commutation superoperator for the inside commutator. Although viable, this is an inefficient way to produce the double commutator superoperator.

1. Ernst, Bodenhausen, and Wokaun, page 20, equation (2.1.59)

ator as can be seen from the original single commutator definition.

$$\begin{aligned}\Gamma 1 \Gamma 2 &= \Gamma 1 \{ (\mathbf{Op}2 \otimes \mathbf{E}) - (\mathbf{E} \otimes \mathbf{Op}2^T) \} \\ \Gamma 1 \Gamma 2 &= \{ (\mathbf{Op}1 \otimes \mathbf{E}) - (\mathbf{E} \otimes \mathbf{Op}1^T) \} \{ (\mathbf{Op}2 \otimes \mathbf{E}) - (\mathbf{E} \otimes \mathbf{Op}2^T) \} \\ \Gamma 1 \Gamma 2 &= (\mathbf{Op}1 \otimes \mathbf{E})(\mathbf{Op}2 \otimes \mathbf{E}) - (\mathbf{Op}1 \otimes \mathbf{E})(\mathbf{E} \otimes \mathbf{Op}2^T) \\ &\quad - (\mathbf{E} \otimes \mathbf{Op}1^T)(\mathbf{Op}2 \otimes \mathbf{E}) + (\mathbf{E} \otimes \mathbf{Op}1^T)(\mathbf{E} \otimes \mathbf{Op}2^T)\end{aligned}$$

Using the distributive property of tensor algebra¹,

$$\Gamma 1 \Gamma 2 = (\mathbf{Op}1 \mathbf{Op}2 \otimes \mathbf{E}) - (\mathbf{Op}1 \otimes \mathbf{Op}2^T) - (\mathbf{Op}2 \otimes \mathbf{Op}1^T) + (\mathbf{E} \otimes \mathbf{Op}1^T \mathbf{Op}2^T) \quad (14-9)$$

Use of equation (14-9) is computationally more efficient than application of equation (14-8) as demonstrated by the following figure.

Mathematical Operations Necessary for Double Commutator

	Equation (11.2)	Equation (11.4)	Equation (11.5)
Op x Op (Hilbert Space)			2
Op +/- Op (Hilbert Space)	1	2	3
Op x Op (Hilbert Space)	2	4	4
LOp x LOp (Liouville Space)		1	

Figure 10-3 - This table shows the mathematical operations used in the three equations applicable to the computation of the double commutation superoperator.

Equation (14-8) involves 2 single commutator calculations (with (14-4)) as well as one superoperator multiplication step. The Liouville space multiplication is computationally long and not necessary when using equation (14-9). Although (14-9) has two operator multiplications and one more operator subtraction than does (14-8) the total computation needed for these should be much less than the superoperator multiplication².

1. The distributive property used here is $(A \otimes B)(C \otimes D) = (AC) \otimes (BD)$.

2. The computational differences will be more pronounced as the Liouville space size increases.

It may be of some use to show the derivation of equation (14-9) pictorially. Considering a single spin 1/2 system, each Hilbert space operator will be a 2x2 array. Equation (14-8) looks schematically as

$$\begin{aligned} \Gamma \mathbf{1} \Gamma \mathbf{2} = & \left(\begin{bmatrix} Op1 \end{bmatrix} \otimes \begin{bmatrix} 1 & \\ & 1 \end{bmatrix} \right) \mathbf{x} \left(\begin{bmatrix} Op2 \end{bmatrix} \otimes \begin{bmatrix} 1 & \\ & 1 \end{bmatrix} \right) \\ & - \left(\begin{bmatrix} Op1 \end{bmatrix} \otimes \begin{bmatrix} 1 & \\ & 1 \end{bmatrix} \right) \mathbf{x} \left(\begin{bmatrix} 1 & \\ & 1 \end{bmatrix} \otimes \begin{bmatrix} Op2^T \end{bmatrix} \right) \\ & - \left(\begin{bmatrix} 1 & \\ & 1 \end{bmatrix} \otimes \begin{bmatrix} Op1^T \end{bmatrix} \right) \mathbf{x} \left(\begin{bmatrix} Op2 \end{bmatrix} \otimes \begin{bmatrix} 1 & \\ & 1 \end{bmatrix} \right) \\ & + \left(\begin{bmatrix} 1 & \\ & 1 \end{bmatrix} \otimes \begin{bmatrix} Op1^T \end{bmatrix} \right) \mathbf{x} \left(\begin{bmatrix} 1 & \\ & 1 \end{bmatrix} \otimes \begin{bmatrix} Op2^T \end{bmatrix} \right) \end{aligned}$$

When the cross products are performed, the matrix size increases (from Hilbert to Liouville space). The picture then appears as the following.

$$\begin{aligned} \Gamma \mathbf{1} \Gamma \mathbf{2} = & \begin{bmatrix} Op1_{11}E & Op1_{12}E \\ Op1_{21}E & Op1_{22}E \end{bmatrix} \mathbf{x} \begin{bmatrix} Op2_{11}E & Op2_{12}E \\ Op2_{21}E & Op2_{22}E \end{bmatrix} - \begin{bmatrix} Op1_{11}E & Op1_{12}E \\ Op1_{21}E & Op1_{22}E \end{bmatrix} \mathbf{x} \begin{bmatrix} Op2^T & 0 \\ 0 & Op2^T \end{bmatrix} \\ & - \begin{bmatrix} Op1^T & 0 \\ 0 & Op1^T \end{bmatrix} \mathbf{x} \begin{bmatrix} Op2_{11}E & Op2_{12}E \\ Op2_{21}E & Op2_{22}E \end{bmatrix} + \begin{bmatrix} Op1^T & 0 \\ 0 & Op1^T \end{bmatrix} \mathbf{x} \begin{bmatrix} Op2^T & 0 \\ 0 & Op2^T \end{bmatrix} \end{aligned}$$

The next step is to perform the sub-matrix multiplications.

$$\Gamma 1 \Gamma 2 = \begin{array}{cc} \begin{array}{cc} (Op1_{11} Op2_{11} + Op1_{12} Op2_{21}) E & (Op1_{11} Op2_{12} Op1_{12} Op2_{22}) E \\ (Op1_{21} Op2_{11} Op1_{22} Op2_{21}) E & (Op1_{21} Op2_{12} Op1_{22} Op2_{22}) E \end{array} & - \begin{array}{cc} Op1_{11} \cdot Op2^T & Op1_{12} \cdot Op2^T \\ Op1_{21} \cdot Op2^T & Op1_{22} \cdot Op2^T \end{array} \\ - \begin{array}{cc} Op2_{11} \cdot Op1^T & Op2_{12} \cdot Op1^T \\ Op2_{21} \cdot Op1^T & Op2_{22} \cdot Op1^T \end{array} & + \begin{array}{cc} Op1^T Op2^T & 0 \\ 0 & Op1^T Op2^T \end{array} \end{array}$$

It is thus obvious how the distributive property applies.

$$\Gamma 1 \Gamma 2 = \left(\begin{array}{|c|} \hline Op1 \\ \hline \end{array} \times \begin{array}{|c|} \hline Op2 \\ \hline \end{array} \otimes \begin{array}{|c|} \hline 1 \\ \hline 1 \end{array} \right) - \begin{array}{|c|} \hline Op1 \\ \hline \end{array} \otimes \begin{array}{|c|} \hline Op2^T \\ \hline \end{array} - \begin{array}{|c|} \hline Op2 \\ \hline \end{array} \otimes \begin{array}{|c|} \hline Op1^T \\ \hline \end{array} + \begin{array}{|c|} \hline 1 \\ \hline 1 \end{array} \otimes \left(\begin{array}{|c|} \hline Op1^T \\ \hline \end{array} \times \begin{array}{|c|} \hline Op2^T \\ \hline \end{array} \right)$$

This is of course equivalent to our working equation (14-9).

$$\Gamma 1 \Gamma 2 = (Op1 Op2 \otimes E) - (Op1 \otimes Op2^T) - (Op2 \otimes Op1^T) + (E \otimes Op1^T Op2^T)$$

Of final note is the situation which occurs when the two operators are equivalent, i.e. when

$$Op = Op1 = Op2.$$

The equation then becomes

$$\Gamma = \Gamma 1 \Gamma 1 = (Op^2 \otimes E) - 2(Op \otimes Op^T) + (E \otimes Op^{T^2}) \quad (14-10)$$

which requires one less cross product to be taken than in equation (14-9).

The Unitary Transformation Superoperator

The unitary transformation superoperator is defined to be¹,

$$\Gamma A = Op A Op^{-1} \quad (14-11)$$

1. Ernst, Bodenhausen, and Wokaun, page 24, equation (2.1.83)

where Γ is a general superoperator (not explicitly a unitary transformation superoperator) and both \mathbf{Op} and \mathbf{A} are general operators. The superoperator in this case relates to the operator \mathbf{Op} according to

$$\Gamma = \mathbf{Op} \otimes \mathbf{Op}^* \quad (14-12)$$

The superscript * implies the complex conjugate and \otimes the tensor product.

Superoperator Exponentials

Whenever superoperators mix with operators it is preferable to perform any basis transformations in the Hilbert space, *i.e.* on the operators not the superoperators. This is simply because of the vast increase in size of the matrices involved. This desire motivated class superoperator maintaining its basis in Hilbert space and not Liouville space.

Seemingly contrary to this, there are times when one desires the superoperator in a different basis and this is in fact allowed. This situation typically arises in computation of the following exponential.

$$\exp[(\mathbf{LOp})t] \quad (14-13)$$

To compute this exponential the superoperator must be placed in its eigenbasis, where the superoperator matrix is diagonal. In this instance, there will be a basis matrix in Liouville space which will convert the eigenbasis (diagonal form) back to the initial basis.

$$(\mathbf{U}_L^{SB})^\dagger \mathbf{LOp}^{EB} \mathbf{U}_L^{SB} = \mathbf{LOp}^{SB} \quad (14-14)$$

Here \mathbf{U}_L is a Liouville space transformation matrix or basis matrix. Superscript SB is used for the original superoperator basis, and superscript EB for the eigenbasis. \mathbf{LOp}^{EB} is the superoperator in its eigenbasis and is diagonal. The exponential now becomes

$$\exp[(\mathbf{LOp}^{SB})t] = (\mathbf{U}_L^{SB})^\dagger \{ \exp[(\mathbf{LOp}^{EB})t] \} \mathbf{U}_L^{SB} . \quad (14-15)$$

When this exponential is to be determined for different times it is essential that the superoperator be stored in its eigenbasis and this mandates the storage of the corresponding basis in Liouville space.

2 Dynamic Spin Systems

2.1 Overview

Class *Dynamic Spin System* (*sys_dynamic*) is provided for the manipulation of. Class *sys_dynamic* is derived from two classes, *spin_system* and *coord_vec*. It is derived from the former so that each dynamic system may function in simulations as if it were a *spin_system*. It is derived from *coord_vec* because each spin can have a defined coordinate associated with it.

2.2 Available Dynamic Spin System Functions

Basic Functions

<code>sys_dynamic</code>	- Dynamic system constructor
<code>=</code>	- Dynamic system assignment
<code>+=</code>	- Dynamic system unary addition

Dynamic Spin System I/O Functions

<code>write</code>	- Write dynamic system to disk file (as a parameter set).
<code>read</code>	- Read dynamic system from disk file (from a parameter set).
<code>print</code>	- Send dynamic system to output stream.
<code><<</code>	- Send dynamic system to an output stream

Dynamic Spin System Correlation Time Functions

<code>taus</code>	- Three correlation times (in seconds).	page 31
<code>taux</code>	- Retrieve/Set x-axis correlation time	page 32
<code>tauy</code>	- Retrieve/Set y-axis correlation time	page 32
<code>tauz</code>	- Retrieve/Set z-axis correlation time	page 32

Dynamic Spin System Chemical Shift Functions

<code>dipoles</code>	- Number of dipoles present in the dynamic spin system.	page 33
<code>xiD_matrix</code>	- Matrix of dipolar interaction constants for spin system	page 33
<code>TD</code>	- Dipolar spatial tensor access.	page 36

Dynamic Spin System Dipole Functions

<code>dipoles</code>	- Number of dipoles present in the dynamic spin system.	page 33
<code>xiD_matrix</code>	- Matrix of dipolar interaction constants for spin system	page 33
<code>TD</code>	- Dipolar spatial tensor access.	page 36

Dynamic Spin System Quadrupole Functions

<code>dipoles</code>	- Number of dipoles present in the dynamic spin system.	page 33
<code>xiD_matrix</code>	- Matrix of dipolar interaction constants for spin system	page 33
<code>TD</code>	- Dipolar spatial tensor access.	page 36

Since *Class sys_dynamic* is derived from the classes *coord_vec* and *spin_system*. Each dynamic system variable maintains all of the abilities of these two classes. Functions appropriate for dynamic system manipulation which are inherited from *coord_vec* and *spin_system* are indicated below. For further details consult the chapter specific for the respective class.

Inherited Spin System Functions (*Classes spin_system & spin_sys*)

spins	- Number of spins	page 224
HS	- Retrieve spin or dynamic spin system Hilbert space	page 224
isotope	- Set or retrieve spin isotope type	page 225
symbol	- Retrieve spin isotope type as a string, e.g. 19F.	page 226
qn	- Retrieve spin angular momentum of a spin or the system.	page 226
element	- Retrieve spin element type as a string, e.g. Carbon.	page 227
momentum	- Retrieve spin angular momentum as a string.	page 227
gamma	- Retrieve gyromagnetic ration of a spin.	page 227
qState	- Retrieve the state vector of a particular quantum state	page 228
qStates	- Retrieve a matrix with a description of all quantum states	page 229
qnState	- Retrieve the quantum number of a state	page 230
qnStates	- Retrieve the quantum number of all states	page 230
qnDist	- Retrieve a statistic over the quantum numbers	page 231
CoherDist	- Retrieve a statistic over the different coherences	page 232
homonuclear	- True or false test whether system is homonuclear.	page 232
heteronuclear	- True or false test whether system is heteronuclear.	page 233
flags	- Set all spin flags to true or false.	page 233
flag	- Set or retrieve individual spin(s) flag true/false status.	page 234
name	- Set or retrieve dynamic spin system name.	page 234
shifts	- Set all chemical shifts to a specific value.	page 247
shift	- Set or retrieve an individual chemical shift (Hertz).	page 248
PPM	- Set or retrieve an individual chemical shift in PPM.	page 251
Js	- Set all coupling constants to a specific value.	page 252
J	- Set or retrieve an individual coupling constant.	page 252
Omega	- Set or retrieve the spectrometer frequency.	page 253
Nyquist	- Retrieve a Nyquist frequency for the system.	page 255
center	- Retrieve a chemical shift center.	page 255

Keep in mind that functions which may be apply to dynamic spin systems are inherited from the base classes. As functions are added to the base classes the automatically are added here (when the GAMMA library is recompiled). Thus there may be other functions in these base classes which are useful for certain applications yet not documented in this chapter. Check the documentation in those classes for further details. Furthermore, when a certain function is desired for the manipulation of a molecule the user should ask whether the function is specific for a molecule or general for one of it's base classes. If the later is true the new function(s) should be added to the base class rather than here. The inheritance will immediately make it applicable here also.

2.3 Correlation Time Functions

2.3.1 taus

Usage:

```
#include <LSLib/sys_dynamic.h>
coord sys_dynamic::taus();
```

Description:

This function returns a coordinate containing the three correlation times for overall spin system motion about the principal axes of diffusion.

Return Value:

A coordinate.

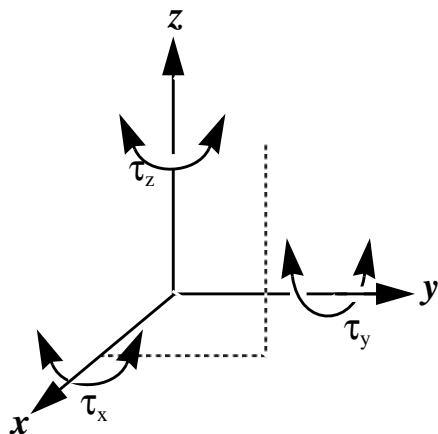
Example:

```
#include <gamma.h>
main()
{
    sys_dynamic sys;                // Set up a dynamic spin system
    sys.read("pfPHE.dsys");         // Read system from disk file
    cout << "\nTaus (nsec):" << sys.taus()*1.e9; // Output the correlation times
}
```

Mathematical Basis:

The figure below illustrates how the three correlation times describe the overall spin system motion.

Three Correlation Times Relative to Diffusion Axes



2.3.2 **taux**

2.3.3 **tauy**

2.3.4 **tauz**

Usage:

```
#include <LSLib/sys_dynamic.h>
double sys_dynamic::taux() const;
double sys_dynamic::tauy() const;
double sys_dynamic::tauz() const;
void sys_dynamic::taux(double tau);
void sys_dynamic::tauy(double tau);
void sys_dynamic::tauz(double tau);
```

Description:

The functions *taux*, *tauy*, and *tauz* are used to either set or obtain the spin system correlation time about the x, y, or z axis respectively. For both input and output the value is in seconds.

Return Value:

A double precision number or nothing.

Example:

```
#include <gamma.h>
main()
{
    sys_dynamic sys;                // Set up a dynamic spin system
    sys.taux(1.e-9);                // Set taux to 1 nanosecond
    sys.tauy(1.e-9);                // Set tauy to 1 nanosecond
    sys.tauz(1.e-9);                // Set tauz to 1 nanosecond
    cout << "\nTaux (nsec):" << sys.taux()*1.e9; // Output taux in nanoseconds
    cout << "\nTaui (nsec):" << sys.taui()*1.e9; // Output taui in nanoseconds
    cout << "\nTautz (nsec):" << sys.tautz()*1.e9; // Output tautz in nanoseconds
}
```

2.4 Dipole Functions

2.4.1 dipoles

Usage:

```
#include <gamma.h>
int sys_dynamic::dipoles();
```

Description:

The function *dipoles* determines the total number of unique dipoles in the dynamic spin system. It is a member function of the class.

Return Value:

An integer is returned.





Example:

```
#include <gamma.h>
main()
{
    sys_dynamic sys;                // Set up a dynamic spin system
    sys.read("pfPHE.dsys");         // Read system from disk file
    cout << "Number of dipoles is " << sys.dipoles(); // Output the number of dipoles
}
```

Mathematical Basis:

It is easy to see how the number of dipoles relates directly to the number of spins by direct examination of lower size spin systems.

Number of Dipoles per Number of Spins

Spins	1	2	3	4	N
Dipoles	0	1	3	6	$\sum_N (N-1)$
Representation					

2.4.2 xiD_matrix

Usage:

```
#include <gamma.h>
matrix sys_dynamic::xiD_matrix();
```

Description:

The function *xiD* returns a matrix containing all dipolar interaction constants for the dynamic spin system as

defined by

$$\xi_{ij}^D = -2 \sqrt{\frac{6\pi}{5}} \left(\frac{\mu_0}{4\pi} \right) h \frac{\gamma_i \gamma_j}{r_{ij}^3} = -2 \sqrt{\frac{6\pi}{5}} DCC_{ij}.$$

Return Value:

An matrix is returned, units will be rad/sec.

Example:

```
#include <gamma.h>
main()
{
    sys_dynamic sys;                // Set up a dynamic spin system
    sys.read("pfPHE.dsys");         // Read system from disk file
    matrix xis = sys.xiD_matrix();   // Get the matrix of xi values
    cout << complex(1.e-6)*xis;     // Output scaled dipolar int. constants.
}
```

Mathematical Basis:

The dipolar interaction constant, ξ_{ij}^D , is a GAMMA defined quantity which sets dipolar Hamiltonians in terms of standardized spatial and spin tensors.

$$\mathbf{H}_{i,j}^D = \xi_{ij}^D \sum_{m=-2}^2 (-1)^m A_{2-m}^D(ij) \mathbf{T}_{2m}^D(ij)$$

It relates directly to the dipolar interaction constant, DCC . The value of ξ_{ij}^D will now be explicitly calculated for two protons 1\AA apart¹. Using the value of Plancks constant to be

$h = 2\pi\hbar = 1.05459 \times 10^{-34} \text{ J}\cdot\text{sec}$ with $\gamma_{\text{proton}} = 2.675 \times 10^8 \text{ sec}^{-1} T^{-1}$ we have

$$\xi_{HH}^D|_{1\text{\AA}} = -2 \sqrt{\frac{6\pi}{5}} \frac{\mu_0}{4\pi} \frac{h\gamma_H\gamma_H}{(1\text{\AA})^3} = -\sqrt{\frac{6\pi}{5}} \frac{\mu_0}{2\pi} \left[\frac{(1.05459 \times 10^{-34} \text{ J}\cdot\text{sec})(2.675 \times 10^8 \text{ sec}^{-1} T^{-1})^2}{10^{-30} \text{ m}^3} \right]$$

$$\xi_{HH}^D|_{1\text{\AA}} = -\sqrt{3.76991} \frac{\mu_0}{2\pi} (1.055 \times 10^{-4} \text{ J}\cdot\text{secm}^{-3})(7.156 \times 10^{16} \text{ sec}^{-2} T^{-2})$$

$$\xi_{HH}^D|_{1\text{\AA}} = -1.942 \frac{\mu_0}{2\pi} (7.546 \times 10^{12} \text{ Jsec}^{-1} \text{m}^{-3} T^{-2}) = \frac{-\mu_0}{2\pi} (1.465 \times 10^{13} \text{ Jsec}^{-1} \text{m}^{-3} T^{-2})$$

1. The GAMMA defined dipolar interactions constant is slightly different from the commonly used dipolar interaction constant. In frequency units, the latter is defined as $\gamma_{ij}^D = \frac{h\gamma_i\gamma_j}{r_{ij}^3}$ (neglecting the factor $\frac{\mu_0}{4\pi}$).

Now substituting in the equalities $\mu_0 = 4\pi \times 10^{-7} \text{J}\cdot\text{sec}^2\text{C}^{-2}\text{m}^{-1}$ and $1\text{T} = 1\text{J}\text{C}^{-1}\text{sec}\cdot\text{m}^{-2}$

$$\xi_{HH}^D \Big|_{1\text{\AA}} = (-2 \times 10^{-7} \text{J}^1 \text{sec}^2 \text{C}^{-2} \text{m}^{-1}) (1.465 \times 10^{13} \text{J}^{-1} \text{C}^2 \text{sec}^{-3} \cdot \text{m})$$

$$\xi_{HH}^D \Big|_{1\text{\AA}} = -2.93 \times 10^6 \text{sec}^{-1}$$

Since all spin isotopes (except tritium) have a smaller gyromagnetic ratio than protons and most atoms will be farther apart than 1\AA , most likely $|\xi_{ij}^D| < 2.93 \times 10^6 \text{sec}^{-1}$. As distance increases this value drops dramatically, for example at 2\AA , $\xi_{HH}^D = -3.664 \times 10^5 \text{sec}^{-1}$.

2.4.3 DCC

Usage:

```
#include <gamma.h>
double sys_dynamic::DCC(int i, int j);
double sys_dynamic::DCC(int i, int j, double nu);
```

Description:

This function *DCC* provides access to spin system dipolar coupling constants according to

$$DCC_{ij} = \frac{\mu_0 h \gamma_i \gamma_j}{4\pi r_{ij}^3}$$

1. *DCC*(int i, int j) - The dynamic spin system furnishes all coupling constant values. In this case the constant for the spin pair *i,j* is returned.
2. *DCC*(int i, int j, double nu) - NOT ACTIVE. This will someday be used to set the dipolar coupling constant (and associated coordinates).

The output units for this function are Hertz. Two values expected from this function would be¹

$$DCC(^1\text{H} - ^1\text{H}) \Big|_{2\text{\AA}} = 9.39 \times 10^4 \text{sec}^{-1} = 14.95 \text{kHz}$$

$$DCC(^1\text{H} - ^{13}\text{C}) \Big|_{1\text{\AA}} = 1.88 \times 10^5 \text{sec}^{-1} = 30 \text{kHz}$$

Return Value:

Either a matrix or a double is returned.

Example:

```
#include <gamma.h>
sys_dynamic dsys;                                     // Set up a dynamic system
dsys.read("filename.dsys");                           // Read in system from file
double D01 = dsys.DCC(0, 1);                          // Get dipolar coupling between 1 & 2
```

1. These were taken from B.C Gerstein and C.R. Dybowski, "Transient Techniques in NMR of Solids", Academic Press, Inc., New York, 1985. Specifically, see Chapter 3, page 94.

2.4.4 TA

Usage:

```
#include <gamma.h>
space_T sys_dynamic::TA(int i, int j);
space_T sys_dynamic::TA(int dip);
```

Description:

The function **TA** returns the dipolar spatial tensor (rank 2) associated with either the two spins indicated in the argument list or the dipole index. A fatal error will result if spin *i* is set to spin *j*.

Return Value:

An spatial tensor is returned.

Example:

```
#include <gamma.h>
main()
{
    sys_dynamic sys;                // Set up a dynamic spin system
    sys.read("pfPHE.dsys");         // Read system from disk file
    space_T SphT;                   // Set an empty spatial tensor
    SphT = sys.TA(0,1);              // SphT dipolar spatial tensor, first 2 spins
}
```

Mathematical Basis:

2.5 Quadrupole Functions

2.5.1 QCC

Usage:

```
#include <gamma.h>
double sys_dynamic::QCC(int i, double nu);
double sys_dynamic::QCC(int i);
```

Description:

This function *QCC* provides access to spin system quadrupolar coupling constants according to

$$DCC_{ij} = \frac{\mu_0}{4\pi} \frac{h\gamma_i\gamma_j}{r_{ij}^3}$$

1. DCC(int i, int j) - The dynamic spin system furnishes all coupling constant values. In this case the constant for the spin pair *i,j* is returned.
2. DCC(int i, int j, double nu) - NOT ACTIVE. This will someday be used to set the dipolar coupling constant (and associated coordinates).

The output units for this function are Hertz. Two values expected from this function would be¹

$$DCC(^1H - ^1H)|_{2A} = 9.39 \times 10^4 \text{sec}^{-1} = 14.95 \text{kHz}$$

$$DCC(^1H - ^{13}C)|_{1A} = 1.88 \times 10^5 \text{sec}^{-1} = 30 \text{kHz}$$

Return Value:

Either a matrix or a double is returned.

Example:

```
#include <gamma.h>
main()
{
    sys_dynamic dsys;                // Set up a dynamic system
    dsys.read("filename.dsys");       // Read in system from file
    double D01 = dsys.DCC(0, 1);      // Get dipolar coupling between 1 & 2
}
```

1. These were taken from B.C Gerstein and C.R. Dybowski, "Transient Techniques in NMR of Solids", Academic Press, Inc., New York, 1985. Specifically, see Chapter 3, page 94.

2.6 Description

Class *sys_dynamic* is implemented so that users can readily manipulate a set of atomic coordinates. Each spin in the dynamic spin system carries isotope label (its spin quantum number), a Cartesian coordinate (e.g. {x,y,z}), and spins specific tensor information (chemical shielding and quadrupole). Furthermore, each spin pair carries applicable tensor information (dipolar and scalar coupling). Overall, three correlation times are carried inside the dynamic spin system to indicate overall motion about the principal axes of motion¹. A right handed Cartesian coordinate system is used to specify where individual spins lie.

Right Handed Cartesian Coordinate System

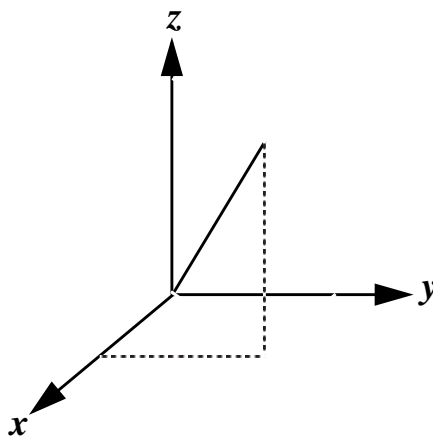


Figure 10-4

1. Assuming rotational diffusion the principal axes of motion will be the principal axes of diffusion. These are typically the same axes about which specifies the three molecular moments of inertia where the spin system is a part of molecule whose motion is dictated by the overall molecular structure. Spin Cartesian coordinates should be entered relative to these principal axes.

2.7 Dynamic System Parameter Files

This section describes how an ASCII file may be constructed that is self readable by a dynamical spin system. The file can be created with any editor and is read with the dynamical system member function “read”. An example of one such file is given in its entirety at the end of this section. 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 dynamical system.

Table 1: Dynamical Spin System Parameters

Parameter	Assumed Units	Examples 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 Iso(1) (2) : 3H - Spin Isotope Type Iso(3) (2) : 13C - Spin Isotope Type
Omega	MHz	Omega (1) : 270.0 - Spectrometer Frequency
CS_T	PPM	CS_T(0) (4) : 2 - Chemical Shift Tensor (111.0, 50.0, -1.5) (0.00, 0.00, 0.00)
v	Hz	v(1) (1) : 2.37 - Chemical Shift (Hz)
PPM	PPM	PPM (1) : 113.2 - Chemical Shift (PPM)
J	Hz	J(0,1) (1) : 2.37 - Coupling Constant (Hz) J(0,3) (1) : 12.1 - Coupling Constant (Hz) J(2,3) (1) : 8.06 - Coupling Constant (Hz)
Coord	Angstroms	Coord(0) (3) : (1.00, 0.00, 0.00) - Coordinate(Ang.)
Taus	nsec	Taus (3) : 2.37 - Correlation Times (nsecs)
Tausp	psec	Tausp (3) : 113.2 - Correlation Times (psecs)
Ds	10^8sec^{-1}	Ds (3) : 1.11 - Diffusion Constants ($10^{**}8/\text{sec}$)
DsHz	10^8Hz	DsHz (3) : 0.024 - Diffusion Constants ($10^{**}8 \text{ Hz}$)
Q_T	Hz	Q_T(0) (4) : 2 - Quadrupolar Tensor (0.0, 50.0, -1.5) (0.00, 0.00, 0.00)

Table 1: Dynamical Spin System Parameters

Parameter	Assumed Units	Examples Parameter (Type) : Value - Statement	
ν	Hz	$\nu(1)$	(1) : 2.37 - Chemical Shift (Hz)
PPM	PPM	PPM	(1) : 113.2 - Chemical Shift (PPM)
Kex(i,j,...)	sec ⁻¹	Kex(0,2)	(1) : 0.5 - 1st & 3rd spins exchange
		Kex(0,1,2)	(1) : 1000.0 - 1st,2nd,3rd spins exchange

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 tagging some output file with the spin system. This parameter is optional.

Table 2: Dynamical System Name

Parameter	Assumed Units	Examples Parameter (Type=2) : Value - Statement	
SysName	none	SysName	(2) : Glycine - Spin System Name

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.

Table 3: Dynamical System NSpins

Parameter	Assumed Units	Examples Parameter (Type=0) : Value - Statement	
NSpins	none	Nspins	(0) : 5 - Number of Spins

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. Isotopes are specified by the atomic number

immediately followed (no blanks) by the atomic symbol..

Table 4: Isotope Types

Parameter	Assumed Units	Examples Parameter (Type=3) : Value - Statement
Iso(i)	none	Iso(0) (3) : 19F - Spin Isotope Type Iso(1) (3) : 3H - Spin Isotope Type Iso(3) (3) : 13C - Spin Isotope Type

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 3 indicates a string parameter.

Spectrometer Frequency: Omega (inherited from class spin_system)

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.

Table 5: Spectrometer Frequency

Parameter	Assumed Units	Examples Parameter (Type=1) : Value - Statement
Omega	MHz	Omega (1) : 270.0 - Spectrometer Frequency

Parameter type 1 indicates a floating point number as the parameter value.

Chemical Shifts: CS_T(i) v(i), PPM(i) (partially inherited from class spin_system)

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..

Table 6: Isotropic Chemical Shifts

Parameter	Assumed Units	Examples Parameter (Type=1) : Value - Statement
CS_T	PPM	CS_T(0) (4) : 2 - Chemical Shift Tensor (111.0, 50.0, -1.5) (0.00, 0.00, 0.00)
v	Hz	Taus (1) : 2.37 - Chemical Shift (Hz)
PPM	PPM	Tausp (1) : 113.2 - Correlation Times (psecs)

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.

Isotropic Scalar Coupling: J(i,j) (inherited from class spin_system)

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.

Table 7: Isotropic Scalar Couplings

Parameter	Assumed Units	Examples Parameter (Type=1) : Value - Statement		
J	Hz	J(0,1)	(3) : 2.37	- Coupling Constant (Hz)
		J(0,3)	(3) : 12.1	- Coupling Constant (Hz)
		J(2,3)	(3) : 8.06	- Coupling Constant (Hz)

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.

Spin Coordinates: Coord

Cartesian spin coordinates Motional parameters indicate to overall movement of the spin system about the diffusion axes. Typically this is specified by three correlation times, $\{\tau_x, \tau_y, \tau_z\}$. For the majority of molecules in liquids, correlation times will range from picoseconds to nanoseconds. An alternative means of specifying the overall motion is with rotational diffusion constants, $\{D_x, D_y, D_z\}$ which have values near 10^8sec^{-1} . To accomodate all of these standards, class *sys_dynamic* will look for the following parameters (in order of priority).

Table 8: Motional Parameters

Parameter	Assumed Units	Examples Parameter (Type=3) : Value - Statement		
Coord	A	Coord(0)	(3) : 2.37	- Spin Coordinate in Angstroms

The motional parameters must be specified, if not the program will stop itself when these are found missing. Internally, GAMMA maintains these parameters as correlation times in SI units (seconds). When these are written by GAMMA to a similar file they will be output as correlation times in nanoseconds. Parameter type 3 indicates a string parameter.

Motional parameters: Taus, Tausp, Ds, DsHz

Motional parameters indicate to overall movement of the spin system about the diffusion axes. Typically this is specified by three correlation times, $\{\tau_x, \tau_y, \tau_z\}$. For the majority of molecules in liquids, correlation times will range from picoseconds to nanoseconds. An alternative means of specifying the overall motion is with rotational diffusion constants, $\{D_x, D_y, D_z\}$ which have values near 10^8sec^{-1} . To accomodate all

of these standards, class *sys_dynamic* will look for the following parameters (in order of priority).

Table 9: Motional Parameters

Parameter	Assumed Units	Examples Parameter (Type=3) : Value - Statement	
Taus	nsec	Taus (1) : 2.37	- Correlation Times (nsecs)
Tausp	psec	Tausp (1) : 113.2	- Correlation Times (psecs)
Ds	10^8sec^{-1}	Ds (1) : 1.11	- Diffusion Constants ($10^{**8}/\text{sec}$)
DsHz	10^8Hz	DsHz (1) : 0.024	- Diffusion Constants (10^{**8}Hz)

The motional parameters must be specified, if not the program will stop itself when these are found missing. Internally, GAMMA maintains these parameters as correlation times in SI units (seconds). When these are written by GAMMA to a similar file they will be output as correlation times in nanoseconds. Parameter type 1 indicates a float parameter.

Quadrupolar parameters: Q_T, QCC, QCCk, QCCM

The quadrupolar nature of a nucleus with $I > 1/2$ can be specified in two primary ways; by definition of the rank 2 quadrupolar tensor or by input of a quadrupolar coupling constant with an optional asymmetry parameter. Quadrupolar coupling constants can be input directly in either Hz, kHz, or MHz. To accomodate all of these standards, class *sys_dynamic* will look for the following parameters (in order of priority).

Table 10: Quadrupolar Parameters

Parameter	Assumed Units	Examples Parameter (Type=4,1) : Value - Statement	
Q_T	PPM	Q_T(0) (4) : 2 (0.0, 50.0, -1.5) (0.00, 90.00, 0.00)	- Quadrupolar Tensor
QCC	Hz	QCC(1) (1) : 113.2	- Quad. Coupling 2nd spin (Hz)
QCCk	kHz	QCCk(0) (1) : 1.11	- Quad. Coupling 1st spin (kHz)
QCCM	MHz	QCCM(3) (1) : 0.024	- Quad. Coupling 4th spin (MHz)
etaQ	none	etaQ(3) (1) : 0.24	- Quad. asymmetry 4th spin

If quadrupolar parameters are not specified they are assumed negligible. Any values specified for a spin with $I = 1/2$ will be ignored. GAMMA will preferentially read a quadrupolar tensor and, if present for a particular spin, subsequently ignores any values QCC and eta values specified for the spin. When written by GAMMA, quadrupolar tensors will be output as tensors if Euler angles exist or QCC, etaQ pairs if no angles have been assigned. Parameter type 1 indicates a floating point parameter and parameter 4 a tensor.

Exchange parameters: Kex

Spins within a dynamic system can be labeled as being involved in mutual exchange¹ processes with param-

eters with a base name of Kex. In such cases, Kex is used to define an exchange process in which two or more spins are exchanging within the system. The value of Kex, input in units of inverted seconds, is the rate at which the exchange process takes place and must be non-negative. The spins involved in a specified mutual exchange process must be of the same isotope type.

Table 11: Mutual Exchange Parameters

Parameter	Assumed Units	Examples	
		Parameter (Type=1) : Value - Statement	
Kex(i,j,k,...)	1/sec	Kex(0,1) (1) : 2	- Exchange rate
		Kex(0,1,3) (1) : 2	- Exchange rate
		Kex(0,1) (1) : 2	- Exchange rate

If quadrupolar parameters are not specified they are assumed negligible. Any values specified for a spin with $I = 1/2$ will be ignored. GAMMA will preferentially read a quadrupolar tensor and, if present for a particular spin, subsequently ignores any values QCC and eta values specified for the spin. When written by GAMMA, quadrupolar tensors will be output as tensors if Euler angles exist or QCC, etaQ pairs if no angles have been assigned. Parameter type 1 indicates a floating point parameter and parameter 4 a tensor.

-
1. A mutual exchange process in one in which no net change to the spin system occurs: the system before exchange is the same as the system after the exchange. An example would be the amide protons (AB) in formamide exchanging as a pi rotation occurs about the amide bond. In contrast, non-mutual exchange processes are those in which there are net changes to the system. This would be the case where a system undergoes a conformational change in which the spins no longer exists in the same environment. Their shifts and scalar couplings differ in the two conformations. A second example of non-mutual exchange would be inter-molecular exchange, for example ATP in exchange with ADP and free phosphate. For the treatment of non-mutual exchange, use the spin system class multi_sys rather than sys_dynamic in your GAMMA programs.