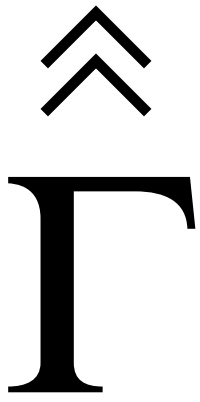
GAMMA

Floquet Module



Author: Dr. Scott A. Smith, Tilo Levante

Date: March 17, 2000

Table of Contents

1 Introduction 3
2 Floquet Operators 4
2.1 Overview 4
2.2 Chapter Contents 4
2.5 F_Operator Basic Functions 9
2.6 F_Operator Complex Functions 13
2.7 F_Operator Internal Access 15
2.9 F_Operator Basis & Representation Manipulations 17
2.10 F_Operator : Floquet Hamiltonian Manipulations 18
3 Floquet2 Operator 24
3.1 Overview 24
3.2 Available F2_Operator Functions 24
3.3 Constructors 26
3.4 F2_Operator Basic Functions 27
3.5 F2_Operator Complex Functions 32
3.6 F2_Operator Internal Access 34
3.7 F2_Operator Basis & Representation Manipulations 35
3.8 F2_Operator: Floquet Hamiltonian Manipulations 36
4 Floquet Theory 39
4.1 Introduction 39
4.2 The Floquet Theorem 39

2 Floquet Operators

2.1 Overview

The class ***Floquet Operator*** defines all the necessary attributes of a Floquet operator **FOp**. The essential components of every Floquet operator are the size of the (truncated) photon space ***N***, the basic Fourier frequency ***omega ,*** and the HiIbert space dimension ***hs*** of an arbitrary operator (defined using class ***gen_op***). The matrix representation of a Floquet Operator **FOp** is then equivalent to the description of a general operator containing (2*N+1)*hs elements. Therefore, a Floquet may further be specified by a matrix mx (see *Class matrix*) and a basis bs (see *Class basis*).

Class *Floquet Operator* includes also specifications of Operator properties (dimension,...), Operator algebras (+, *,...), and definitions of all available Operator functions (exp, prop, ..). Functions are also provided which allow the user direct access into the matrix representation of each **FOp**. Moreover, routines for the correct builtup of a Floquet Hamiltonian (e.g. add_omega) are available.

To use the class *floq_op* it is necessary to include the file floq_op.h.

2.2 Chapter Contents

2.2.1 Section Listing

Overview page 2-4
Chapter Contents page 2-4
Available Functions page 2-6
Constructors and Assignment page 2-7
Basic Functions page 2-9
Complex Functions page 2-14
Internal Access page 2-16
Overview page 2-20
Basis Manipulations page 2-18
Hamiltonian Manipulations page 2-19
Description page 2-20
Floquet Examples page 2-21

2.2.2 Function Listing

Basic Functions

floq_op - Construction page 2-7
= - Assignment page 2-7
+ - Addition page 2-7
+= - Unary Addition page 2-7
- - Assignment page 2-7
-= - Assignment page 2-7
* - Assignment page 2-7

* - Assignment page 2-7

Complex Functions

phodim - trace in photon space page 2-14
size - Operator size page 2-14
hsdim - Hilbert space dimension page 2-14
phodim - photon space dimension page 2-14
omega - Fourier frequency page 2-15
phodim - exponential page 2-14
prop - propagator page 2-15

Internal Access

phodim - Retrieve element page 2-14
phodim - Retrieve FOp block page 2-14
phodim - Assign FOp block page 2-14
phodim - Assign side diagonal page 2-14
phodim - Assign element page 2-14
phodim - Retrieve element page 2-14
phodim - GARP-1 Composite Pulse page 2-14

Basis Manipulations

set_DBR - Set default basis page 2-18
set_EBR - Set eigenbasis page 2-18

Basis Manipulations

print - Output GARP definitions page 6-116
<< - Output GARP definitions page 6-117

2.2.3 Figures & Tables Listing

Figure 6-1 - Basic GARP 25 Step Sequence page 6-123
Figure 6-3 - Reading GARP Parameters page 6-127
Figure 6-4 - 13C Decoupled Spectrum Using GARP-1 page 6-129
Figure 6-5 - 13C Coupled Spectrum, Zero Strength GARP-1 page 6-129
Figure 6-6 - 13C GARP-1 Decoupling Versus RF-Field Strength page 6-131

2.2.4 Examples

Reading GARP Parameters page 6-127
Magic Angle Spinning (I=1/2) page 2-21
GARP-1 Decoupling vs. Field page 6-130
GARP-1 Decoupling Profile page 6-132

2.2.5 Programs

GarpWF0.cc Generate Plot of GARP-1 25 Step Sequence page 6-140

2.3 Available Functions

Basic Functions

floq_op	- Constructor	FOp1, FOp1 (N, hs, omega, mx1), FOp1 (N, hs, omega, mx1, bs), FOp2(FOp1)
=	- Assignment	FOp = FOp1
+	- Addition	FOp1 + FOp2, FOp + mx, mx+FOp
+=	- Addition	FOp += FOp1, FOp += mx
-	- Subtraction	FOp1 - FOp2, FOp - mx, mx - Fop
-=	- Subtraction	FOp -= FOp1, FOp -= mx
*	- Multiplication	FOp1*FOp2, FOp * mx, mx * FOp
*=	- Multiplication	FOp*= FOp1, FOp *= mx1
/	- Division	FOp1/z
/=	- Division	FOp/=z
&=	- Reverse Multiplication	FOp&= FOp (FOp = FOp1 * FOp), FOp &= mx (FOp=mx * FOp)page 12
<<	- Send floq_op to output stream	

floq_op: Floquet Hamilton-Manipulations

add_omega	- Add omegas on main diag.	FOp.add_omega()
sub_omega	- Subtract omegas	FOp.sub_omega()

2.4 Constructors and Assignment

2.4.1 floq_op

Usage:

```
floq_op ();
floq_op (int N, int hs, double omega);
floq_op (int N, int hs, double omega, matrix& mx1);
floq_op(int N, int hs, double omega ,matrix& mx1, basis& bs1);
```

Description:

The function ***floq_op*** is used to create a Floquet operator.

1. floq_op() - Creates an “empty” NULL Floquet operator. Can be later filled by an assignment.
2. floq_op(int N, int hs, double omega): sets up an floq_op with the truncated photon dimension N, the Hilbert space dimension hs and the Fourier frequency omega.
3. floq_op (int N, int hs, double omega, matrix mx1): creates a floq_op with mx1 as the floq_op representation. The matrix dimension has to be equal to (2*N+1)*hs, where N and hs again represent photon and Hilbert space,respectively.
4. floq_op (int N, int hs, double omega, matrix mx1, basis bs1): With a matrix mx1 and a basis bs1 as arguments, the function sets up an floq_op with matrix representation mx1 in the basis bs1 which must have the same dimension size. The basis must relate properly to the default basis, meaning that the basis transformation matrix can be used to transform mx1 into the default basis (see *Class Basis*).The matrix dimension has to be equal to (2*N+1)*hs, where N and hs represent photon and Hilbert space,respectively. The Fourier frequency is denoted omega.
5. floq_op (const floq_op& FOp1): Finally, one may produce an floq_op from another floq_op. The new floq_op is equivalent to the current representation of the input floq_op **FOp1**.
6. Return Value:
7. A new floq_op which may be subsequently used with all defined floq_op functions.

Return Value:

floq_op returns no parameters. It is used strictly to create a Floquet operator.

Examples:

```
include <gamma.h>
main ()
  {
  PulGARP PG;
  PulGARP PG1(538.9, “13C”);
  PulGARP PG3(PG1);
  }
```

See Also: =

2.4.2 =

Usage:

```
void floq_op::floq_op = (const floq_op& FOp1);
```

Description:

This allows for the ability to assign an floq_op to another floq_op. For the assignment of two floq_ops **FOp** = **FOp1**, floq_op **FOp** is set equal to floq_op **FOp1** exclusively in the working basis of floq_op **FOp1**.

Note:

Keep in mind that the assignment floq_op = like the binaries +, -, * and / works only on one floq_op representation, i.e. the formula **FOp** = **FOp1** produces the floq_op **FOp** in a single representation (in the basis of floq_op **FOp1**) regardless of how many stored representations of **FOp1** exist.

2.5 Basic Functions

2.5.1 +

Usage:

```
floq_op floq_op + (floq_op& FOp1, floq_op& FOp2);
floq_op floq_op + (floq_op& FOp1,matrix& mx1);
floq_op floq_op + (matrix& mx1, floq_op& FOp1);
```

Description:

FOp1 + FOp2: adds two floq_ops **FOp1** and **FOp2**.

FOp1 + mx1: adds a matrix mx1 to an floq_op **FOp1**.

mx1 + FOp1: definition of the addition of an floq_op **FOp1** to a matrix mx1. The result is equivalent to the previous addition, it produces a new floq_op in the default basis.

Note:

FOp1 + FOp2: a check is made to insure that both floq_ops are in the same basis. If this is not true, floq_op **FOp2** is transformed into the basis of floq_op **FOp1** prior to the addition, thus insuring that the addition produces a result in (and only in) the same basis of **FOp1**.Moreover, it is tested, whether the matrix representation of both floq_ops has the same dimension.

FOp1 + mx1: the matrix mx1 is assumed to be a matrix in the default basis and the addition takes place in the default basis. floq_op **FOp1** is first placed in the default basis, the addition takes place, and then the result is a new floq_op in the default basis.

The binary floq_op + inherently works on only one floq_op representation, i.e. the formula **FOp3** = **FOp1** + **FOp2** produces the floq_op **FOp3** in a single representation (in the basis of floq_op **FOp1**) regardless of how many stored representations of **FOp1** and/or **FOp2** exist. Since GAMMA will transform **FOp3** into any needed basis automatically, this should present no limitations while keeping computation time and memory usage down. One should keep in mind that a consequence of this is that the FOperation **FOp2**= **FOp2** + **FOp1** will set any current representations of floq_op **FOp2** to zero except the result representation.

This applies to all binary FOperations +, -, * and / with floq_ops and superfloq_ops.

Return Value:

A new floq_op which exists in an appropriate representation.

2.5.2 +=

Usage:

```
void floq_op::floq_op += (matrix& mx1);
```

Description:

The assignment floq_op += is used to handle the addition left = left + term. Left stands for the current input floq_op **FOp** and term is a matrix.

Note:

A check is made to insure that both floq_opar and matrix mx are in the same basis and have the same dimension. If necessary, floq_op **FOp1** is transformed into the basis of floq_op **FOp** first to guarantee that the result is in the same basis of **FOp**.

Return Value:

A new floq_op which exists in an appropriate representation.

2.5.3 -

Usage:

```
floq_op floq_op - (floq_op& FOp1, floq_op& FOp2);
floq_op floq_op - (floq_op& FOp1, matrix& mx1);
floq_op floq_op - (matrix& mx1, floq_op& FOp1);
floq_op floq_op- (floq_op& FOp1);
```

Description:

FOp1 - FOp2: subtracts two floq_ops **FOp1** and **FOp2**.

FOp1 - mx1: subtracts a matrix mx1 from an floq_op **FOp1**.

mx1 - FOp1: Definition of the subtraction of an floq_op **FOp1** from a matrix mx1. The result is equivalent to the negative of the previous subtraction, it produces a new floq_op in the default basis.

- FOp1: Definition of the negation of floq_op **FOp1**. The result is an floq_op only in the working basis of **FOp1** which is -1.0 * **FOp1**.

Note:

See function + in this Chapter.

Return Value:

A new floq_op which exists in an appropriate representation

2.5.4 -=

Usage:

```
void floq_op::floq_op -= (floq_op& FOp1);
void floq_op::floq_op -= (matrix& mx1);
```

Description:

The assignment floq_op -= is used to handle the subtraction left = left - term. Term can be either another floq_op or a matrix.

Note:

See function += in this Chapter.

2.5.5 *

Usage:

```
floq_op floq_op * (floq_op& FOp1, floq_op& FOp2);
floq_op floq_op * (floq_op& FOp1, matrix& mx1);
floq_op floq_op * (matrix& mx1, floq_op& FOp1);
floq_op floq_op * (complex& z1, floq_op& FOp1);
floq_op floq_op * (floq_op& FOp1, complex& z1);
floq_op floq_op * (floq_op& FOp1, double d);
floq_op floq_op * (double d, floq_op& FOp1);
```

Description:

This allows for the multiplication of two floq_ops **FOp1** and **FOp2**, the multiplication of an floq_op and a matrix, and for the multiplication of a scalar and an floq_op.

For the multiplication of a scalar with an floq_op, the scalar is multiplied into each element of **FOp1** to produce an floq_op in the same basis of **FOp1.** The multiplication of an floq_op with a scalar produces the same result as multiplication of a scalar with an floq_op. The scalar can be either a complex number z1 or a double d.

Note:

Since floq_ops are represented by matrices the result of the multiplication of two floq_ops or an floq_op with a matrix depends on the succession of the included arguments.

For further explanations see function + in this Chapter.

2.5.6 *=

Usage:

```
void floq_op::floq_op *= (floq_op& FOp1);
void floq_op::floq_op *= (matrix& mx1);
void floq_op::floq_op *= (complex& z1);
```

Description:

The assignment floq_op *= is used to handle the multiplication left = left*term. The term can be an floq_op **FOp1**, a matrix mx1 or a complex number z1. Left denotes the current input floq_op **FOp**.

Note:

See function += in this Chapter.

2.5.7 /

Usage:

floq_op floq_op / (floq_op& FOp1, complex& z1);

Description:

Divides each matrix element of the included floq_op **FOp1** by a complex number z1.

Return Value:

A new floq_op in the WBR of **FOp1**.

2.5.8 /=

Usage:

void floq_op::floq_op /= (complex& z1);

Description:

The assignment floq_op /= is used to handle the division left = left/z1. This is performed exclusively in the working basis of current floq_op. The scalar can be either a complex number z1 or a double d.

Note:

See function += in this Chapter.

2.5.9 &=

Usage:

void floq_op::floq_op& = (floq_op& FOp1);
void floq_op::floq_op& = (matrix& mx1);

Description:

Manages the multiplication of the current floq_op **FOp** with another floq_op **FOp1** or a matrix mx1 in the order **FOp** = term* **FOp**. The result is exclusively in the DBR.

2.5.10 <<

Usage:

friend ostream& floq_op << (ostream& ostr, const floq_op& FOp1);

Description:

Puts the included floq_op to the output stream ostr. In addition, photon space and Hilbert space dimension and the Fourier frequency omega are displayed.

Return Value:

The modified output stream ostr.

2.6 Complex Functions

2.6.1 pho_trace

Usage:

```
gen_op pho_trace(floq_op& FOp1);
```

Description:

Calculates the trace over all (-N,..+N) photon space elements, that belong to the same Hilbert space element. The result is Hilbert space operator gen_op.

Return Value:

A gen_op in the WBR of **FOp1**.

2.6.2 size

Usage:

```
int floq_op::size();
```

Description:

Calculates the size of **FOp1**.

Return Value:

An integer number that is the size of **FOp1**.

2.6.3 hsdim

Usage:

```
int floq_op::hsdim();
```

Description:

The function hsdim returns the floq_ops Hilbert space dimension hs.

Return Value:

An Integer.

2.6.4 phodim

Usage:

```
int floq_op::phodim();
```

Description:

The function phodim returns the floq_ops photon space dimension (see *Class Spin Sys* on page 196) N.

Return Value:

An Integer.

2.6.5 omega

Usage:

```
int floq_op::omega();
```

Description:

The function omega returns the floq_ops Fourier frequency omega. (see *Class Spin Sys* on page 196) omega.

Return Value:

A double.

2.6.6 exp

Usage:

```
floq_op exp (floq_op& FOp1);
```

Description:

Calculates the exponential of **FOp1**. This is done in the eigenbasis of the input floq_op.

Return Value:

The floq_op in the working basis of **FOp1**.

2.6.7 prop

Usage:

```
floq_op prop (floq_op& Fham, double time);
```

Description:

Calculates the propagator of floq_op **Fham** according to $\exp(-2\pi i(time)(Fham))$.

Return Value:

A new floq_op consisting out of the propagator of **FOp1**.

2.7 Internal Access

2.7.1 ()

2.7.2 get_block

2.7.3 put_block

Usage:

```
gen_op floq_op::floq_op () (int N1, int N2);
gen_op floq_op::get_block (int N1, int N2);
void floq_op::put_block (gen_op& Op, int N1, int N2);
```

Description:

Gets or sets the Hilbert space operator Op at position (N1,N2) (defined as photon space indices) in the WBR of the FOp. When getting the operator,

$$\boldsymbol{Op} = \langle N1|\boldsymbol{FOp}|N2\rangle \tag{2-8}$$

that returned is found at row N1, column N2 where $N1, N2 \in [-N,N]$.

Note:

In Gamma the numbering of the matrix elements starts with zero to row - 1 respectively column - 1 according to the notation in C and C++.

Return Value:

Either void or a general operator in Hilbert space.

2.8.4 put_sdiag

Usage:

```
void floq_op::put_sdiag(gen_op& Op1, int sdn);
```

Description:

Returns a Floquet operator, where the side diagonal number sdn has been entirely filled with the general operator Op1 (with dimension hs). sdn = +1,-1 specifies the train on the right side and left of the main diagonal, respectively.

Return Value:

A floq_op with the sidediagonal number sdn filled with operator Op1.

2.8.5 put

Usage:

```
void floq_op::put (complex& z1, int N1, int N2, int H1, int H2);
void floq_op::put (complex& z1, int row, int col)
```

Description:

Sets the matrix element specified with the photon space indices (N1,N2) and Hilbert space indices (H1,H2) (or using the row and column (col) numbers) the current floq_op to the included complex number z1. This is performed in the WBR.

2.8.6 get

Usage:

```
complex floq_op::get ( int N1, int N2, int H1, int H2);
complex floq_op::get (int row, int col)
```

Description:

Returns the matrix element specified with the photon space indices (N1,N2) and Hilbert space indices (H1,H2) (or using the row and column (col) numbers) in the current floq_op. This is performed in the WBR.

2.9 Basis Manipulations

2.9.1 set_DBR

Usage:

```
void floq_op::set_DBR ();
```

Description:

The function set_DBR insures that the input floq_op is currently in its default basis representation. If the default basis representation of this floq_op is not internally maintained it will be computed by similar transformation and then stored (within the confines of imposed limits set for the representations of **FOp**). **FOp** will then have its working basis set to its default basis.

2.9.2 set_EBR

Usage:

```
void floq_op::set_EBR ();
```

Description:

The function set_EBR insures that the current input floq_op **FOp** is currently in its eigenbasis representation. If the eigenbasis representation of **FOp** is not internally maintained it will be computed by matrix diagonalization and then stored (within the confines of imposed limits set for the representations of **FOp**). **FOp** will then have its working basis set to its eigenbasis.

2.10 Hamiltonian Manipulations

2.10.1 add_omega

Usage:

```
void floq_op::add_omega ();
```

Description:

The function add_omega insures the correct builtup of a Floquet Hamiltonian by adding multiples of the basic Fourier frequency omega to the values on the main diagonal of an arbitrary Floquet operator **FOp**. Numbering the diagonal position from k = -N to +N (see e.g. 10.6.3) where N is the photon dimension of **FOp**, k*omega***1** is added to each Hilbert space operator. **1** denotes the unity operator in the Hilbert space defined by hs. The calculation is performed in the working basis of **FOp.**

Return Value:

A floq_op with multiples of omega added to the main diagonal values.

2.10.2 sub_omega

Usage:

```
void floq_op::sub_omega ();
```

Description:

The function sub_omega insures the correct builtup of a Floquet Hamiltonian by subtracting multiples of the basic Fourier frequency omega from the values on the main diagonal of an arbitrary Floquet operator **FOp**. Numbering the diagonal position from k = -N to +N (see e.g. 10.6.3) where N is the photon dimension of **FOp**, k*omega***1** is subtracted from each Hilbert space operator. **1** denotes the unity operator in the Hilbert space defined by hs. The calculation is performed in the working basis of **FOp.**

Return Value:

A floq_op with multiples of omega subtracted from the main diagonal values.

2.11 Description

2.12 Floquet Examples

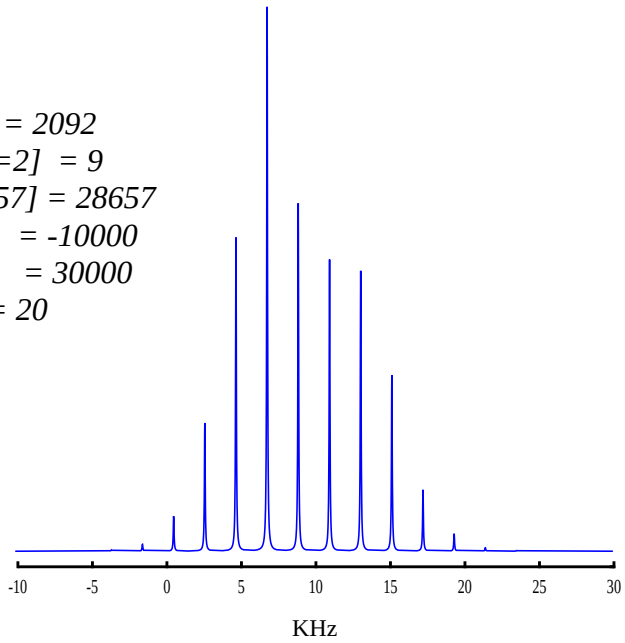
2.12.1 Magic Angle Spinning (I=1/2)

In this section we shall produce a simple 1D NMR spectrum of a single spin 1/2 nucleus under MAS. The periodic time dependence in the spatial coordinates will be simulated using the Floquet formalism. Depending on the size of the chemical shift anisotropy (CSA), MAS sidebands will occur.

The program will simulate the MAS powder pattern of a single spin system (I=1/2) under the influence of CSA (Chemical Shielding Anisotropy.) The time dependence is dscribed using the Floquet formalism. A Floquet Hamiltonian is generated for each powder crystallite orientation. Each is then used to generate a spectrum & the spectra summed over all orientations with the appropriate weighting. The Cheng method is used for the powder averaging sheme. The spectrum is output to an ASCII file compatible with Gnuplot. Gnuplot is called at the end of the program to display the spectrum on screen.

Simulated Proton MAS Spectrum

sigmaxx [Hz] = 1850
sigmayy [Hz] = 7600
sigmazz [Hz] = 17410
rotation frequency [Hz] = 2092
Floquet dimension N,[N>=2] = 9
 # powder orient.[max 28657] = 28657
Minimal frequency [Hz] = -10000
Maximal frequency [Hz] = 30000
Line Broadening = 20



The spectrum above was generated by running the program. The input (dialog as the program runs) is shown to to the left. As one can see, first one specifies the principal components of the shielding tensor (σ_{xx}, σ_{yy}, σ_{zz}). Next the rotor frequency is input followed by the Floquet dimension to use in the treatment. Next comes a specification of how well to do the powder average. Finally the output plotting limits are set and a paramter to broaden each crystallite spectrum (spectrum smoothing). The program source is on the following page.

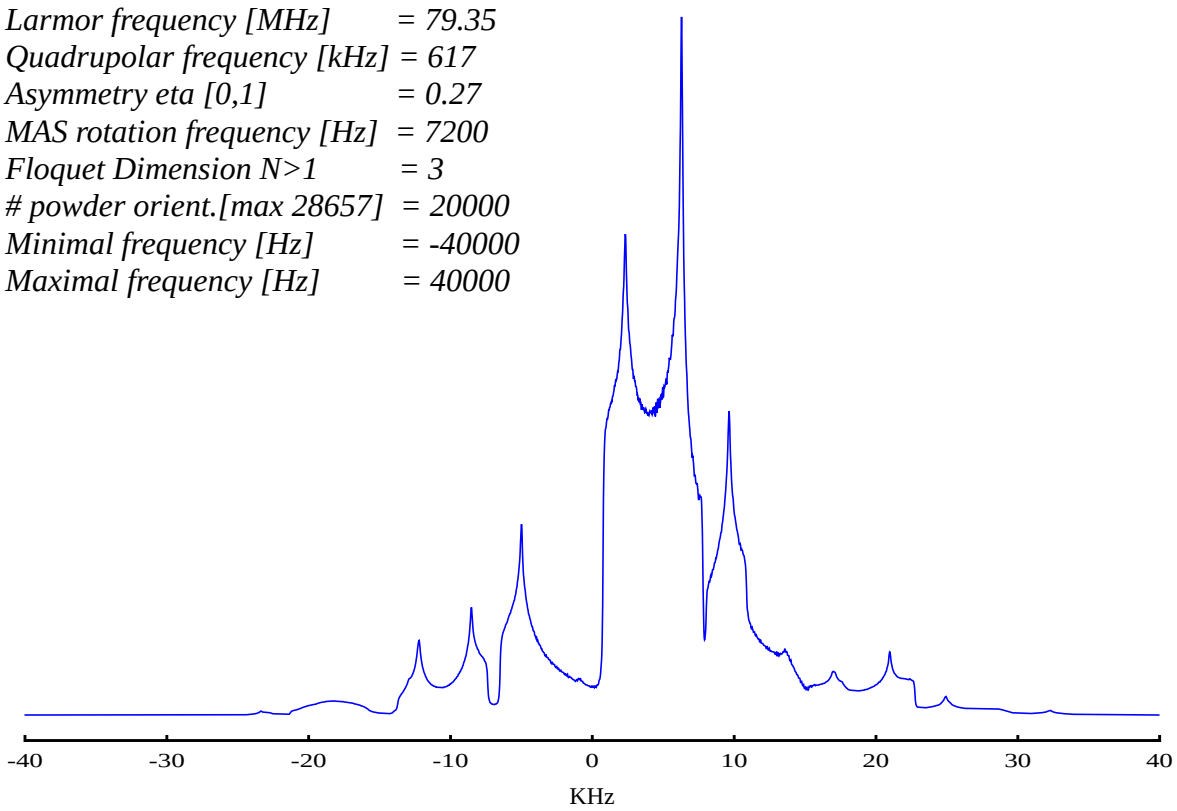
```
/* FloqMAS0.cc ***************************************************************-*-c++-*-
**                                                                              **
** This program simulates the MAS powder pattern of a 1 spin system            **
** (I=1/2) under the influence of CSA (Chemical Shielding Anisotropy.)         **
** The time dependence is dscribed using the Floquet formalism.                **
**                                                                              **
*********************************************************************************/
#include <gamma.h>
int main (int argc, char *argv[])
 {
//              First Get Cartesian PAS values & The Spherical Angles
  int qn = 1;                                   // Parameter query index
  double sixx,siyy,sizz;                        // SA PAS: sigmaxx,sigmayy,sigmazz
  query_parameter(argc,argv,qn++,"sigmaxx [Hz]    = ", sixx);
  query_parameter(argc,argv,qn++,"sigmayy [Hz]    = ", siyy);
  query_parameter(argc,argv,qn++,"sigmazz [Hz]    = ", sizz);
/       Need rotation speed, Floquet dimension, # Steps in Powder Average
  int N;                                        // Floquet dimension
  double omegar;                                // MAS rotation frequency
  int steps;                                    // Steps in powder loop
  query_parameter (argc,argv,qn++,"rotation frequency [Hz]     = ", omegar);
  query_parameter (argc,argv,qn++,"Floquet dimension N,[N>=2]  = ", N);
  query_parameter (argc,argv,qn++,"# powder orient.[max 28657] = ", steps);
//          Need Plotting Range & Line Broadening To Smooth Spectrum
  double Fmin, Fmax;                            // Spectral range
  int lb;                                       // Line broadening parameter
  query_parameter (argc,argv,qn++,"Minimal frequency  [Hz]     = ", Fmin);
  query_parameter (argc,argv,qn++,"Maximal frequency  [Hz]     = ", Fmax);
  query_parameter (argc,argv,qn++,"Line Broadening             = ", lb);
//                  Set Up Internal Variables For the Calculation
  spin_system sys(1);                           // A single spin (I=1/2) system
  coord B(0,0,1);                               // Bo field vector (along +z)
  matrix s1(3,3,complex0);                      // A 3x3 array for SA spatial tensor
  s1.put_h(sixx, 0, 0);                         // Set <0|SA|0> to be sigmaxx
  s1.put_h(siyy, 1, 1);                         // Set <1|SA|1> to be sigmayy
  s1.put_h(sizz, 2, 2);                         // Set <2|SA|2> to be sigmazz
  space_T CS_pas(A2(s1));                       // Cast this as a spatial tensor
  spin_T TTS = T_CS2(sys,0,B);                  // This is the SA spin tensor
  space_T CS;                                   // For oriented SA spatial tensor
  gen_op D = Fm(sys);                           // Set up a detection operator
  int hs = sys.HS();                            // System spin Hilbert space
//              Set Up The System At Point Following A Hard 90y Pulse
  gen_op sigma0=sigma_eq(sys);                  // Set system to equilibrium
  gen_op sigma1=lypuls(sys,sigma0,90.);         // Apply a hard 90y pulse
  floq_op fsigma(N, hs, omegar);                // Floquet operator
  fsigma.put_block(sigma1, 0, 0);               // Floquet system after 90y pulse
  gen_op H_0 = CS_pas.component(0,0)            // Space/Time indepenent part of
             * TTS.component(0,0);              // Hamiltonian (A20*T20)
  gen_op H_1, H_2;                              // For Hamiltonian Fourier expansion
  int NP = 4096;                                // Acquisition block size
  row_vector spect(NP), specsum(NP);            // Set data blocks for use powder loop
  double phi,theta;                             // Rotation angles in powder loop
  int CN = 28657;                               // Value for Cheng average
//                      Start of Powder Averaging
  for(int b=1;b<CN;b=b+int(CN/steps))           // Powder loop over theta and phi
   {                                            // using Cheng's method
   cout << "Cheng b Value: "<< b  << " of " << CN // Tell user where we are at in
        << " Until Powder Average End" << "\r"; // the simulation
   theta=180./double(CN)*b;                     // Crystal theta in [0,180]
   phi=double(360./CN *((10946*b)%CN));         // Crystal phi   in [0,360]
   CS = CS_pas.rotate(phi,theta,0.);            // Reorient spatial tensor
/*                      Build The Floquet Hamiltonian
                 +            +
          -2i*PI*t(2*H2 )  -2i*PI*t(H1 )          2i*PI*t(H1)   2i*PI*t(2*H2)
  H = e                + e              + H0  + e           + e                    */
    H_1 = CS.component(2,1)                     // Space/Time dependent parts of H
      * TTS.component(2,0);                     //       C1*(A21*T20)
   H_1 *= (1/sqrt(3.));
   H_2 = CS.component(2,2)                      //       C2*(A22*T20)
       * TTS.component(2,0);
   H_2 *= (1/sqrt(6.));
   floq_op HF(N, hs, omegar);                   // Hamilton Floquet Matrix
   HF.put_sdiag(adjoint(H_2),-2);               // Set side diagonal # -2
   HF.put_sdiag(adjoint(H_1),-1);               // Set side diagonal # -1
   HF.put_sdiag(H_0,0);                         // Set main diagonal #  0
   HF.put_sdiag(H_1,1);                         // Set side diagonal #  1
   HF.put_sdiag(H_2,2);                         // Set side diagonal #  2
   HF.add_omega();                              // Add omegas on diagonal

// Calculate spectrum & sum with other crystallite spectra assuming the
// third axis for the powder average is the same as the MAS spinning axis

   spec_maspowder(fsigma,D,HF,Fmin,Fmax,NP,spect);
   spect *= sin(theta*PI/180.);
   specsum += spect;
   }
//          Looping Done, Smooth The Spectrum & Output To Screen
  specsum = IFFT(specsum);                      // Put spectrum in time domain
  exponential_multiply(specsum,-lb);            // Apodize for smoothing
  specsum = FFT(specsum);                       // FFT back to frequency domain
  GP_1D("mas.asc",specsum,0,Fmin,Fmax);         // Output plot in Gnuplot ASCII
  GP_1Dplot("mas.gnu", "mas.asc");              // Interactively plot
  cout << "\n\n";                               // Keep screen nice
 }
```

2.12.2 Magic Angle Spinning (Quadrupolar)

The MAS powder pattern of the central transition (1/2 -> -1/2) of a quadrupolar Hamiltonian is simulated. The calculation is based on the results of secular average Hamiltonian theory. The MAS time dependence is parametrically introduced and described using the Floquet formalism.

The program will simulate the MAS powder pattern of a single spin system (23Na, I=3/2) under the influence of the Zeeman and Quadrupolar interacitons. The time dependence is dscribed using the Floquet formalism. A Floquet Hamiltonian is generated for each powder crystallite orientation. Each is then used to generate a spectrum & the spectra summed over all orientations with the appropriate weighting. The Cheng method is used for the powder averaging sheme. The spectrum is output to an ASCII file compatible with Gnuplot. Gnuplot is called at the end of the program to display the spectrum on screen.

Simulated 23Na MAS Spectrum



The spectrum above was generated by running the program. The input (dialog as the program runs) is shown to to the left. As one can see, first one specifies the Larmor frequecy (Ω) and the spherical principal components of the quadrupolar tensor (QCC, η). Next the rotor frequency is input followed by the Floquet dimension to use in the treatment. Next comes a specification of how well to do the powder average. Finally the output plotting limits are set. The program source code follows.

```
/* FloqQMAS0.cc ************************************************************-*-c++-*-
*/
#include <gamma.h>
int main (int argc, char*argv[])
 {
//                          Input Simulation Parameters
 int qn=1;                                  // Parameter query index
 string outFileName;                        // Output filename
 double omegal;                             // Larmor frequency
 double omegaq;                             // Quadrupolar frequency
 double omegar;                             // Rotation (=MAS) frequency
 double eta;                                // Quad tensor asymmetry
 int steps;                                 // Steps in powder loop
 int N;                                     // Floquet dimension
 double Fmin, Fmax;                         // Spectral range
 query_parameter(argc,argv,qn++,"Larmor frequency [MHz]       = ", omegal);
 query_parameter(argc,argv,qn++,"Quadrupolar frequency [kHz]  = ", omegaq);
 query_parameter(argc,argv,qn++,"Asymmetry eta [0,1]          = ", eta);
 query_parameter(argc,argv,qn++,"MAS rotation frequency [Hz]  = ", omegar);
 query_parameter(argc,argv,qn++,"Floquet Dimension N>1        = ", N);
 query_parameter(argc,argv,qn++,"# powder orient.[max 28657]  = ", steps);
 query_parameter(argc,argv,qn++,"Minimal frequency [Hz]       = ", Fmin);
 query_parameter(argc,argv,qn++,"Maximal frequency [Hz]       = ", Fmax);
//                    Set Up Internal Variables For the Calculation
 spin_system sys(1);                        // Set up a single spin system
 sys.isotope (0,"23Na");                    // Set it to 23Na (I=3/2)
 double Vxx,Vyy,Vzz;                        // AQ PAS: Vxx, Vyy, Vzz
 Vzz = sqrt(omegaq*omegaq/(omegal));        // Convert the spherical PAS
 Vxx = (eta+1)/(-2.)*Vzz;                   // components into Cartesian
 Vyy = (eta-1)/2.*Vzz;                      // values
 matrix Qmx(3, 3, complex0);                // Array for 3x3 QA Cartesian
 Qmx.put_h(Vxx,0,0);                        // which will be diagonal
 Qmx.put_h(Vyy,1,1);                        // in the PAS
 Qmx.put_h(Vzz,2,2);
 space_T AQ_pas = A2(Qmx);                  // Cast into a spatial tensor
 space_T AQ;                                // For rotated spatial tensor
 spin_T  TTQ = T_Q(sys,0);                  // Quadrupolar spin tensor
 int hs = sys.HS();                         // System spin Hilbert space
//                 Set A Detection Operator For the Central Transition
 matrix cen_Trans(hs,hs,0.);                // Begin with an empty operator
 cen_Trans.put(0.5,hs/2-1,hs/2);            // Set for transition 1/2 <=> -1/2
 gen_op D(cen_Trans);                       // Cast as an operator
//              Set Up The System At Point Following A Hard 90y Pulse
 gen_op sigma0 = sigma_eq(sys);             // Equilibrium density operator
 gen_op sigma1 = lypuls(sys,sigma0,90.);    // Apply a 90y pulse to system
 floq_op fsigma(N, hs, omegar);             // Floquet density operator
 fsigma.put_block(sigma1,0,0);              // Set for after 90y pulse
 spin_op S1 = TTQ.component(2,1)*TTQ.component(2,-1)
         - TTQ.component(2,-1)*TTQ.component(2,1);
 spin_op S2 = TTQ.component(2,2)*TTQ.component(2,-2)
         - TTQ.component(2,-2)*TTQ.component(2,2);
 int NP = 4096;                             // Acquisition block size
 row_vector spec(NP);                       // Set data blocks for use
 row_vector specsum(NP);                    // in the powder loop
 double phi,theta;                          // Rotation angles in powder loop
 int CN = 28657;                            // Value for Cheng average
 double CNd = (double)CN;                   // (Value as a double needed too)
 double beta1 = 54.73561;                   // The magic angle
//                          Start of Powder Averaging
 for(int b=1;b<CN;b=b+int(CN/steps))        //Powder loop over theta and phi
 {
 cout << "Cheng b Value: "<< b << " of "     // Tell user where we are at in
     << CN << " Until Powder Average End"   // the simulation
     << "\r";
 theta=180./CNd*b;                          // Crystal theta in [0,180]
 phi=double(360./CN*((10946*b)%CN));        // Crystal phi   in [0,360]
 AQ = AQ_pas.rotate(phi,theta,0.);          // Rotate spatial tensor
//                        Build The Floquet Hamiltonian
 gen_op H[10];
 for(int i=0; i<=4; i++)
  for(int j=0;j<=4;j++)
    H[i+j] += (S1*d2(1,i-2,beta1)*d2(-1,j-2,beta1)
          * AQ.component(2,i-2)*AQ.component(2,j-2))
          + (0.5*S2*d2(-2,i-2,beta1)*d2(2,j-2,beta1)
          * AQ.component(2,i-2)*AQ.component(2,j-2));
 floq_op HF(N, hs, omegar);                 // Set up Floquet Hamiltonian
 HF.put_sdiag(H[4],0);
 HF.put_sdiag(H[5],1);
 HF.put_sdiag(adjoint(H[5]),-1);
 HF.put_sdiag(H[6],2);
 HF.put_sdiag(adjoint(H[6]),-2);
 HF.put_sdiag(H[7],3);
 HF.put_sdiag(adjoint(H[7]),-3);
 HF.put_sdiag(H[8],4);
 HF.put_sdiag(adjoint(H[8]),-4);
 HF.add_omega();
//     Calculate spectrum & sum with other crystallite spectra assuming the
 spec_maspowder(fsigma,D,HF,Fmin,Fmax,NP,spec);// aquisition
 spec=IFFT(spec);                           // Set spectrum in time domain
 specsum += sin(theta*PI/180)*spec;         // Sum up weighted spectra
 }
//      Looping Done, Smooth The Spectrum & Output To Screen
exponential_multiply(specsum,-15);          // Apodize for smoothing
specsum = FFT(specsum);                     // FFT back to frequency domain
GP_1D("qmas.asc",specsum,0,Fmin,Fmax);      // Output plot in Gnuplot ASCII
GP_1Dplot("qmas.gnu", "qmas.asc");          // Interactively plot
```

```
}
```

3 Floquet2 Operator

3.1 Overview

The class *Floquet 2 Operator* defines all the necessary attributes of a (two mode) Floquet Operator **F2_Op**. The essential components of every two mode Floquet Operator are the sizes of the (truncated) photon spaces (N1,N2), the basic Fourier frequencies (omega1,omega2) and the HiIbert space dimension hs of an arbitrary Operator, that may be defined using class *gen_Op* (see *Class gen_Op*). The matrix representation of a two mode Floquet Operator **F2_Op** is then equivalent to the description of a general operator containing (2*N1+1)*(2*N2+1)*hs elements. Therefore, a two mode Floquet operator may further be specified by a matrix mx (see *Class matrix*) and a basis bs (see *Class basis*).

Class *Floquet 2 Operator* includes also specifications of Operator properties (e.g. dimensions), Operator algebras (+, *,...), and definitions of all available Operator functions (e.g. exp). Functions are also provided which allow the user direct access into the matrix representation of each **F2_Op**. Moreover, routines for the correct builtup of a two mode Floquet Hamiltonian (e.g. add_omegas) are available

To use the class *F2_Operator* it is necessary to include the file floq2_op.h.

3.2 Available F2_Operator Functions

F2_Operator Basic Functions

floq2_op	- Constructor	F2_Op1, F2_Op1 (N1,N2, hs, omega1,omega2, mx1), F2_Op1 (N1,N2, hs, omega1,omega2, mx1, bs), F2_Op2(F2_Op1)
=	- Assignment	F2_Op = F2_Op1
+	- Addition	F2_Op1 + F2_Op2, F2_Op + mx, mx+F2_Op page 95
+=	- Addition	F2_Op += F2_Op1, F2_Op += mx
-	- Subtraction	F2_Op1 - F2_Op2, F2_Op - mx, mx - F2_op page 29
-=	- Subtraction	F2_Op -= F2_Op1, F2_Op -= mx
*	- Multiplication	F2_Op1*F2_Op2, F2_Op * mx, mx * F2_Op
*=	- Multiplication	F2_Op*= F2_Op1, F2_Op *= mx1
/	- Division	F2_Op

&=	- Reverse Multiplication	F2_Op&= F2_Op (F2_Op = F2_Op1 * F2_Op), F2_Op &= mx (F2_Op=mx * F2_Op)page 31
=	-set equal	F2_Op2=F2_Op1
<<	- Send F2_Operator to output stream	

F2_Operator Complex Functions

size	- F2_Operator size	F2_Op.size();
hsdim	- F2_Op, Hilbert space dim.	F2_Op.hsdim()
phodim1	- F2_Op, Photon space dim. 1	F2_Op.phodim1();
phodim2	- F2_Op, Photon space dim. 2	F2_Op.phodim2();
omega1	- F2_Op, Fourier freq. omega1	F2_Op.omega1();
omega2	- F2_Op, Fourier freq. omega2	F2_Op.omega2();
exp	- F2_Operator exponential	exp(F2_Op)

F2_Operator Internal Access

put_block	- Assign F2_Operator block	F2_Op.put_block(Op1,N1x,N1y,N2x,N2y);
put_sdiag	- Assign sidediagonal	F2_Op.put_sdiag(Op1, sdn1,sdn2);

F2_Operator Basis & Representation Manipulations

set_DBR	- Put into default basis	F2_Op.set_DBR()
set_EBR	- Put into eigenbasis	F2_Op.set_EBR()

F2_Operator: Floquet Hamilton-Manipulations

add_omegas	- Add omegas on main diag.	F2_Op.add_omegas()
sub_omegas	- Subtract omegas on main diag.	F2_Op.sub_omegas()

3.3 Constructors

3.3.1 F2_Op

Usage:

floq2_op ();

floq2_op (int N1, int N2, int hs, double omega1, double omega2);

floq2_op (int N1, int N2, int hs, double omega1, double omega2, matrix& mx1);

floq2_op(int N1, int N2, int hs, double omega1, double omega2, ,matrix& mx1, basis& bs1);

Description:

The function floq2_op is used to create an F2_Operator quantity.

floq2_op(): sets up an empty F2_Operator which can be explicitly specified later.

floq2_op(int N1, int N2, , int hs, double omega1, double omega2): sets up an F2_Operator with the truncated photon dimensions N1 and N2, the Hilbert space dimension hs and the Fourier frequencies omega1 and omega2 refering to the photon dimensions N1 and N2,respectively.

floq2_op (int N1,int N2, int hs, double omega1, double omega2, matrix mx1): creates a F2_Operator with mx1 as the F2_Operator representation. The matrix dimension has to be equal to (2*N1+1)*(2*N2+1)*hs, where N1,N2 and hs again represent photon and Hilbert space dimensions,respectively.

floq2_op (int N1, int N2, int hs, double omega1, double omega2, matrix mx1, basis bs1): With a matrix mx1 and a basis bs1 as arguments, the function sets up an F2_Operator with matrix representation mx1 in the basis bs1 which must have the same dimension size. The basis must relate properly to the default basis, meaning that the basis transformation matrix can be used to transform mx1 into the default basis (see *Class Basis* on page 84).The matrix dimension has to be equal to (2*N1+1)*(2*N2+1)*hs, where N1,N2 and hs represent photon and Hilbert space dimensions,respectively. The Fourier frequencies are denoted omega1 and omega2.

floq2_op (const floq2_op& F2_Op1): Finally, one may produce an F2_Operator from another F2_Operator. The new F2_Operator is equivalent to the current representation of the input F2_Operator **F2_Op1**.

Return Value:

A new F2_Operator which may be subsequently used with all defined F2_Operator functions.

3.3.2 floq2_op=

Usage:

void floq2_op::F2_Operator = (const floq2_op& F2_Op1);

Description:

This allows for the ability to assign an F2_Operator to another F2_Operator. For the assignment of two F2_Operators **F2_Op** = **F2_Op1**, F2_Operator **F2_Op** is set equal to F2_Operator **F2_Op1** exclusively in the working basis of F2_Operator **F2_Op1**.

Note:

Keep in mind that the assignment F2_Operator = like the binaries +, -, * and / works only on one F2_Operator representation, i.e. the formula **F2_Op** = **F2_Op1** produces the F2_Operator **F2_Op** in a single representation (in the basis of F2_Operator **F2_Op1**) regardless of how many stored representations of **F2_Op1** exist.

3.4 F2_Operator Basic Functions

3.4.1 +

Usage:

floq2_op F2_Operator + (floq2_op& F2_Op1, floq2_op& F2_Op2);

floq2_op F2_Operator + (floq2_op& F2_Op1,matrix& mx1);

floq2_op F2_Operator + (matrix& mx1, floq2_op& F2_Op1);

Description:

F2_Op1 + F2_Op2: adds two F2_Operators **F2_Op1** and **F2_Op2**.

F2_Op1 + mx1: adds a matrix mx1 to an F2_Operator **F2_Op1**.

mx1 + F2_Op1: definition of the addition of an F2_Operator **F2_Op1** to a matrix mx1. The result is equivalent to the previous addition, it produces a new F2_Operator in the default basis.

Note:

F2_Op1 + F2_Op2: a check is made to insure that both F2_Operators are in the same basis. If this is not true, F2_Operator **F2_Op2** is transformed into the basis of F2_Operator **F2_Op1** prior to the addition, thus insuring that the addition produces a result in (and only in) the same basis of **F2_Op1**.Moreover, it is tested, whether the matrix representation of both F2_Operators has the same dimension.

F2_Op1 + mx1: the matrix mx1 is assumed to be a matrix in the default basis and the addition takes place in the default basis. F2_Operator **F2_Op1** is first placed in the default basis, the addition takes place, and then the result is a new F2_Operator in the default basis.

The binary F2_Operator + inherently works on only one F2_Operator representation, i.e. the formula **F2_Op3** = **F2_Op1** + **F2_Op2** produces the F2_Operator **F2_Op3** in a single representation (in the basis of F2_Operator **F2_Op1**) regardless of how many stored representations of **F2_Op1** and/or **F2_Op2** exist. Since GAMMA will transform **F2_Op3** into any needed basis automatically, this should present no limitations while keeping computation time and memory usage down. One should keep in mind that a consequence of this is that the F2_Operation **F2_Op2**= **F2_Op2** + **F2_Op1** will set any current representations of F2_Operator **F2_Op2** to zero except the result representation.

This applies to all binary F2_Operations +, -, * and / with F2_Operators and superF2_Operators.

Return Value:

A new F2_Operator which exists in an appropriate representation.

3.4.2 +=

Usage:

void floq2_op::F2_Operator += (matrix& mx1);

Description:

The assignment F2_Operator += is used to handle the addition left = left + term. Left stands for the current input F2_Operator **F2_Op** and term is a matrix.

Note:

A check is made to insure that both F2_Operatorar and matrix mx are in the same basis and have the same dimension. If necessary, F2_Operator **F2_Op1** is transformed into the basis of F2_Operator **F2_Op** first to guarantee that the result is in the same basis of **F2_Op**.

Return Value:

A new F2_Operator which exists in an appropriate representation.

3.4.3 -

Usage:

floq2_op F2_Operator - (floq2_op& F2_Op1, floq2_op& F2_Op2);

floq2_op F2_Operator - (floq2_op& F2_Op1, matrix& mx1);

floq2_op F2_Operator - (matrix& mx1, floq2_op& F2_Op1);

floq2_op F2_Operator- (floq2_op& F2_Op1);

Description:

F2_Op1 - F2_Op2: subtracts two F2_Operators **F2_Op1** and **F2_Op2**.

F2_Op1 - mx1: subtracts a matrix mx1 from an F2_Operator **F2_Op1**.

mx1 - F2_Op1: Definition of the subtraction of an F2_Operator **F2_Op1** from a matrix mx1. The result is equivalent to the negative of the previous subtraction, it produces a new F2_Operator in the default basis.

- F2_Op1: Definition of the negation of F2_Operator **F2_Op1**. The result is an F2_Operator only in the working basis of **F2_Op1** which is -1.0 * **F2_Op1**.

Note:

See function + in this Chapter.

Return Value:

A new F2_Operator which exists in an appropriate representation

3.4.4 -=

Usage:

void floq2_op::F2_Operator -= (floq2_op& F2_Op1);

void floq2_op::F2_Operator -= (matrix& mx1);

Description:

The assignment F2_Operator -= is used to handle the subtraction left = left - term. Term can be either another F2_Operator or a matrix.

Note:

See function += in this Chapter.

3.4.5 *

Usage:

floq2_op F2_Operator * (floq2_op& F2_Op1, floq2_op& F2_Op2);

floq2_op F2_Operator * (floq2_op& F2_Op1, matrix& mx1);

floq2_op F2_Operator * (matrix& mx1, floq2_op& F2_Op1);

floq2_op F2_Operator * (complex& z1, floq2_op& F2_Op1);

floq2_op F2_Operator * (floq2_op& F2_Op1, complex& z1);

floq2_op F2_Operator * (floq2_op& F2_Op1, double d);

floq2_op F2_Operator * (double d, floq2_op& F2_Op1);

Description:

This allows for the multiplication of two F2_Operators **F2_Op1** and **F2_Op2**, the multiplication of an F2_Operator and a matrix, and for the multiplication of a scalar and an F2_Operator.

For the multiplication of a scalar with an F2_Operator, the scalar is multiplied into each element of **F2_Op1** to produce an F2_Operator in the same basis of **F2_Op1.** The multiplication of an F2_Operator with a scalar produces the same result as multiplication of a scalar with an F2_Operator. The scalar can be either a complex number z1 or a double d.

Note:

Since F2_Operators are represented by matrices the result of the multiplication of two F2_Operators or an F2_Operator with a matrix depends on the succession of the included arguments.

For further explanations see function + in this Chapter.

3.4.6 *=

Usage:

void floq2_op::F2_Operator *= (floq2_op& F2_Op1);

void floq2_op::F2_Operator *= (matrix& mx1);

void floq2_op::F2_Operator *= (complex& z1);

Description:

The assignment F2_Operator *= is used to handle the multiplication left = left*term. The term can be an F2_Operator **F2_Op1**, a matrix mx1 or a complex number z1. Left denotes the current input F2_Operator **F2_Op**.

Note:

See function += in this Chapter.

3.4.7 /

Usage:

floq2_op F2_Operator / (floq2_op& F2_Op1, complex& z1);

Description:

Divides each matrix element of the included F2_Operator **F2_Op1** by a complex number z1.

Return Value:

A new F2_Operator in the WBR of **F2_Op1**.

3.4.8 /=

Usage:

void floq2_op::F2_Operator /= (complex& z1);

Description:

The assignment F2_Operator /= is used to handle the division left = left/z1. This is performed exclusively in the working basis of current F2_Operator. The scalar can be either a complex number z1 or a double d.

Note:

See function += in this Chapter.

3.4.9 &=

Usage:

void floq2_op::F2_Operator& = (floq2_op& F2_Op1);

void floq2_op::F2_Operator& = (matrix& mx1);

Description:

Manages the multiplication of the current F2_Operator **F2_Op** with another F2_Operator **F2_Op1** or a matrix mx1 in the order **F2_Op** = term* **F2_Op**. The result is exclusively in the DBR.

3.4.10 <<

Usage:

friend ostream& F2_Operator << (ostream& ostr, const floq2_op& F2_Op1);

Description:

Puts the included F2_Operator to the output stream ostr. In addition, photon subspaces N1 and N2 and Hilbert space dimension and the Fourier frequencies omega1,omega2 are displayed.

Return Value:

The modified output stream ostr.

3.5 F2_Operator Complex Functions

3.5.1 size

Usage:

int floq2_op::size();

Description:

Calculates the size of **F2_Op1**.

Return Value:

An integer number that is the size of **F2_Op1**.

3.5.2 hsdim

Usage:

int floq2_op::hsdim();

Description:

The function hsdim returns the F2_Operators Hilbert space dimension (see *Class Spin Sys* on page 196) hs.

Return Value:

An Integer.

3.5.3 phodim1

Usage:

int floq2_op::phodim1();

Description:

The function phodim1 returns the F2_Operators photon space dimension N1 corresponding to the basic Fourier frequency omega1

Return Value:

An Integer.

3.5.4 phodim2

Usage:

int floq2_op::phodim2();

Description:

The function phodim2 returns the F2_Operators photon space dimension N2 corresponding to the basic Fourier frequency omega2 .

Return Value:

An Integer.

3.5.5 omega1

Usage:

int floq2_op::omega1();

Description:

The function omega1 returns the F2_Operators Fourier frequency omega1.

Return Value:

A double.

3.5.6 omega2

Usage:

int floq2_op::omega2();

Description:

The function omega2 returns the F2_Operators Fourier frequency omega2.

Return Value:

A double.

3.5.7 exp

Usage:

floq2_op exp (floq2_op& F2_Op1);

Description:

Calculates the exponential of **F2_Op1**. This is done in the eigenbasis of the input F2_Operator.

Return Value:

The F2_Operator in the working basis of **F2_Op1**.

3.6 F2_Operator Internal Access

3.6.1 put_block

Usage:

void floq2_op::put_block (gen_op& Op1, int N1x, int N1y, int N2x, int N2y);

Description:

Places the Hilbert space operator Op1 at position (N1x, N1y, N2x,N2y) (defined as photon space indices) in the WBR of the F2_Operator.

Return Value:

A F2_Operator with an new block at (N1x,N1y,N2x,N2y).

3.6.2 put_sdiag

Usage:

void floq2_op::put_sdiag(gen_op& Op1, int sdn1, int sdn2);

Description:

Returns a two mode Floquet operator, where the side diagonal specified by the numbers sdn1 and sdn2 has been entirely filled with the general operator Op1 (with dimension hs). sdn1 and sdn2 refer to the Fourier components of omega1 and omega2, respectively.

Return Value:

A F2_Operator with the sidediagonal number set (sdn1,sdn2) filled with operator Op1.

3.7 F2_Operator Basis & Representation Manipulations

3.7.1 set_DBR

Usage:

void floq2_op::set_DBR ();

Description:

The function set_DBR insures that the input F2_Operator is currently in its default basis representation. If the default basis representation of this F2_Operator is not internally maintained it will be computed by similar transformation and then stored (within the confines of imposed limits set for the representations of **F2_Op**). **F2_Op** will then have its working basis set to its default basis.

3.7.2 set_EBR

Usage:

void floq2_op::set_EBR ();

Description:

The function set_EBR insures that the current input F2_Operator **F2_Op** is currently in its eigenbasis representation. If the eigenbasis representation of **F2_Op** is not internally maintained it will be computed by matrix diagonalization and then stored (within the confines of imposed limits set for the representations of **F2_Op**). **F2_Op** will then have its working basis set to its eigenbasis

3.8 F2_Operator: Floquet Hamiltonian Manipulations

3.8.1 add_omegas

Usage:

void floq2_op::add_omegas ();

The function add_omegas insures the correct builtup of a Floquet Hamiltonian by adding multiples of the basic Fourier frequencies omega1 and omega2 to the values on the main diagonal of an arbitrary two mode Floquet operator **F2_Op**. Numbering the diagonal position from k = -N1 to +N1 and l=-N2 to +N2 where N1,N2 are the photon dimensions of **F2_Op**, [(k*omega1)+(l*omega2)] * **1** is added to each Hilbert space operator. **1** denotes the unity operator in the Hilbert space defined by hs. The calculation is performed in the working basis of **F2_Op.**

Return Value:

A F2_Operator with multiples of omega1 and omega2 added to the main diagonal values.

3.8.2 sub_omegas

Usage:

void floq2_op::sub_omegas ();

The function sub_omegas insures the correct builtup of a Floquet Hamiltonian by subtracting multiples of the basic Fourier frequencies omega1 and omega2 from the values on the main diagonal of an arbitrary two mode Floquet operator **F2_Op**. Numbering the diagonal positions from k = -N1 to +N1 and l=-N2 to +N2 where N1,N2 are the photon dimensions of **F2_Op**, [(k*omega1)+(l*omega2)] * **1** is subtracted from each Hilbert space operator. **1** denotes the unity operator in the Hilbert space defined by hs. The calculation is performed in the working basis of **F2_Op.**

Return Value:

A F2_Operator with multiples of omega1 and omega2 subtracted from the main diagonal values.

3.1 Examples

3.1.1 DOR (Quadrupolar)

The nonsychronized DOR (Double Rotation) - powder pattern of the central transition 1/2 -> -1/2 of a quadrupolar Hamiltonian is simulated. The calculation is based on the results of secular average Hamiltonian theory. The periodic time dependence due to DOR is parametrically introduced and described using a two mode Floquet approach.
The following simulation parameters are used:

GAMMA program:	*dor_quad.cc*
output filename*:*	*outFileName*
minimal and maximal frequency defining the spectral range:	*minFreq, maxFreq*
Larmor frequency of considered nucleus (here ^{23}Na) in MHz:	*omegal*
Quadrupolar coupling constant in kHz, (see e.g. Ref. [1,2]):	*omegaq*
Quadrupolar asymmetry parameter η:	*eta*
inner rotor spinning frequency (i.e. Fourier frequency 1) in Hz:	*omegar1*
outer rotor spinning frequency (i.e. Fourier frequency 2) in Hz:	*omegar2*
Size of the martix representation of the Floquet Hamiltonian in dim. 1:	*N1*
Size of the martix representation of the Floquet Hamiltonian in dim. 2:	*N2*
Number of powder orientations, maximal 28657:	*steps*

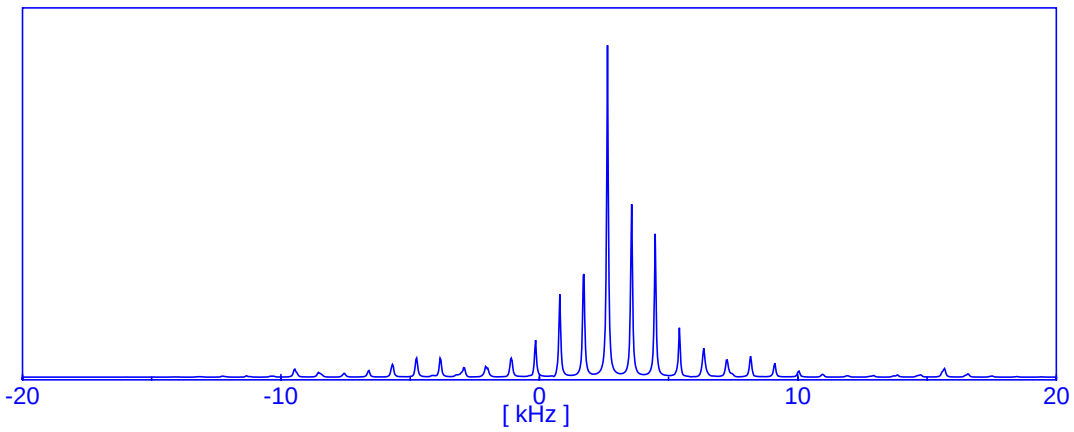
Input parameters:

file	= dor
N	= 9
minfreq,maxfreq	= -60000,+60000
omegal	= 132.29
omegaq	= 617
eta,	= 0.27
omegar1	= 5470
omegar2	= 925
N1	= 2
N2	= 6
steps	= 200

Output parameters:

file	= dor.dat

The resulting output “dor.dat” contains Ascii-format and has here been processed using xvgr.



References:
[1] M. Baldus, T.O. Levante, and B.H. Meier, Z. Naturforsch. 49 (1994) 80 - 88
[2] T.O. Levante, M. Baldus, B.H. Meier, and R.R. Ernst, Mol. Phys. (1995) in press

4 Floquet Theory

4.1 Introduction

The class *Floquet Operator* defines all the necessary attributes of a Floquet Operator. The essential components of every Floquet Operator are the size of the (truncated) photon space N, the basic Fourier frequency omega and the Hilbert space dimension hs of an arbitrary Operator, that may be defined using class *gen_Op* (see *Class gen_Op*). The matrix representation of a Floquet

To use the class *F_Operator* it is necessary to include the file floq_op.h.

4.2 The Floquet Theorem

4.2.1 Time Dependent Schrodinger Equation, Periodic Hamiltonian

We begin with the time-dependent Schrodinger equation

$$ih\frac{d}{dt}|\Psi(t)\rangle = H(t)|\Psi(t)\rangle \tag{0-1}$$

Both the state vector $|\Psi(t)\rangle$ and the Hamiltonian reside in the system Hilbert space of dimension N. If we have an initial solution to the Schrodinger equation, $|\Psi(t_0)\rangle$, the equation is a 1st order differential equation having a unique solution

$$ih\frac{d}{dt}|\Psi(t)\rangle = H(t)|\Psi(t)\rangle \qquad |\Psi(t_0)\rangle = |\Psi_0\rangle \tag{0-2}$$

At a later time τ, these same equations are

$$ih\frac{d}{dt}|\Psi(t+\tau)\rangle = H(t+\tau)|\Psi(t+\tau)\rangle \qquad |\Psi(t_0+\tau)\rangle = |\Psi_\tau\rangle \tag{0-3}$$

If the Hamiltonian is periodic in time τ,

$$H(t+\tau) = H(t) \tag{0-4}$$

and the previous equation becomes

$$ih\frac{d}{dt}|\Psi(t+\tau)\rangle = H(t)|\Psi(t+\tau)\rangle \qquad |\Psi(t_0+\tau)\rangle = |\Psi_\tau\rangle \tag{0-5}$$

We will immediately “simplify” our nomenclature by using a superscript τ to indicate a vector or operator in which the time variable have been incremented by the length of time τ. Thus, we define $|\Psi(t+\tau)\rangle = |\Psi^{\tau}(t)\rangle$, $H(t+\tau) = H(t) = H^{\tau}(t)$, and obtain a time-dependent Schrodinger equation which appears similar to our previous form

$$ih\frac{d}{dt}|\Psi^{\tau}(t)\rangle = H(t)|\Psi^{\tau}(t)\rangle \qquad |\Psi^{\tau}(t_0)\rangle = |\Psi_0^{\tau}\rangle \tag{0-6}$$

4.2.2 Time Evolution Propagator (Hilbert Space)

At this point we have only written two forms of the same differential equation and we wish to obtain sollutions for them. A solution of the original equation can be written in the form

$$|\Psi(t)\rangle = \sum_{n=1}^{N} c_n|\varphi_n(t)\rangle \tag{0-7}$$

where $\{|\varphi_n(t)\rangle\}$ are a set of N particular solutions for the specific time t. By forming the matrix $X(t)$ where

$$X(t) = \left[|\varphi_1(t)\rangle\ |\varphi_2(t)\rangle\ \ldots\ |\varphi_n(t)\rangle\right] \tag{0-8}$$

the time-dependent Schrodinger equation can be written as

$$ih\frac{d}{dt}X(t) = H(t)X(t) \tag{0-9}$$

Note that each individual vector from which $X(t)$ is constructed, $|\varphi_i(t)\rangle$, is time dependent, but can be related to a particular set of time independent basis functions, $\{|\varepsilon_m\rangle\}$, also spanning the Hilbert space.

$$|\varphi_i(t)\rangle = \sum_{m=1}^{N} \varphi_i^m(t)|\varepsilon_m\rangle \tag{0-10}$$

The matrix $X(t)$ can be represented as

$$X(t) = \begin{bmatrix} \varphi_1^1(t) & \varphi_2^1(t) & \ldots & \varphi_N^1(t) \\ \varphi_1^2(t) & \varphi_2^2(t) & \ldots & \varphi_N^2(t) \\ \cdot & \cdot & \cdots & \cdot \\ \cdot & \cdot & \cdots & \cdot \\ \cdot & \cdot & \cdots & \cdot \\ \varphi_1^N(t) & \varphi_2^N(t) & \ldots & \varphi_N^N(t) \end{bmatrix} \tag{0-11}$$

A similar treatment can be done on the equations shifted by the Hamiltonian periodicity time τ. Essentially

$$|\Psi^T(t)\rangle = \sum_{n=1}^{N} d_n|\varphi_n(t)\rangle \tag{0-12}$$

$$ih\frac{d}{dt}X^\tau(t) = H(t)X^\tau(t) \tag{0-13}$$

It is evident that both $X(t)$ and $X^\tau(t)$ satisfy the same time-dependent differential equation. As such, we can surmize that the two X matrices are related through some constant array and we will call that constant matrix R

$$X^\tau(t) = X(t)R \tag{0-14}$$

$$X^\tau(t) = \begin{bmatrix} \varphi_1^1(t) & \varphi_2^1(t) & \dots & \varphi_N^1(t) \\ \varphi_1^2(t) & \varphi_2^2(t) & \dots & \varphi_N^2(t) \\ . & . & \dots & . \\ . & . & \dots & . \\ . & . & \dots & . \\ \varphi_1^N(t) & \varphi_2^N(t) & \dots & \varphi_N^N(t) \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & \dots & r_{13} \\ r_{21} & r_{22} & \dots & r_{23} \\ . & . & \dots & . \\ . & . & \dots & . \\ . & . & \dots & . \\ r_{N1} & r_{N2} & \dots & r_{NN} \end{bmatrix} \tag{0-15}$$

Since the Hamiltonian is Hermitian it can be shown that both the X arrays and R are as well.

$$[X^\tau(t)]^\dagger = [X^\tau(t)]^{-1} \qquad R^\dagger = R^{-1} \tag{0-16}$$

If we let S be the matrix which diagonalizes R (R is Hermitian so the eigenvalues are real), then

$$\Lambda = S^{-1}RS \tag{0-17}$$

and we can relate R to a diagonal exponential matrix Q and the time period τ with

$$\Lambda = S^{-1}RS = e^{-iQ\tau} \tag{0-18}$$

Both Λ and Q are diagonal. Using this relationship we have

$$X^\tau(t) = X(t)R = X(t)Se^{-iQ\tau}S^{-1} \tag{0-19}$$

and thus

$$X(t+\tau)Se^{iQ(t+\tau)} = X^\tau(t)Se^{iQ(t+\tau)} = X(t)Se^{iQt} \tag{0-20}$$

Notice how we now have a nice relationship between the X arrays involving the two times which differ by the period of the Hamiltonian. Defining the operators $Y(t)$ and $Z(t)$ as

$$Y(t) = X(t)S = Z(t)e^{-iQ\tau} \qquad Z(t) = X(t)Se^{iQt} = Z(t+\tau) \tag{0-21}$$

The previous equations become

$$X^{\tau}(t) = Y(t)e^{-iQ\tau}S^{-1} \qquad X^{\tau}(t)Se^{iQ(t+\tau)} = Y(t)e^{iQt} \tag{0-22}$$

We can utilize this to develop the Floquet Theorem. This states that the operator $Y(t)$ is a solution to the differential equation

$$ih\frac{d}{dt}X(t) = H(t)X(t) \tag{0-23}$$

We can show this explicitly. Differentiation of $Y(t)$ with respect to time and multiplication with ih procedes as follows.

$$ih\frac{d}{dt}Y(t) = ih\frac{d}{dt}[X(t)S] = [H(t)X(t)S] = H(t)Y(t) \tag{0-24}$$

4.2.3 Time Evolution Propagator (Hilbert Space)

We can construct an evolution operator (propagator) from time t_0 to time t, it is given by

$$U(t_0, t) = X(t)X^{-1}(t_0) \tag{0-25}$$

where

$$U(t_0, t_0) = I \qquad U(t_0, t)|\Psi_0\rangle = |\Psi(t)\rangle \qquad U(t_0, t)X(t_0) = X(t) \tag{0-26}$$

Under the Floquet treatment we shall express this as

$$U(t_0, t) = F(t)F^{-1}(t_0) \tag{0-27}$$

where

$$F(t) = \Phi(t)e^{-iQt} \qquad \Phi(t+\tau) = \Phi(t) \qquad ih\frac{d}{dt}F(t) = H(t)F(t) \tag{0-28}$$

Using this same reasoning, we can formulate similar equations at the time that is advanced by the period τ

$$ih\frac{d}{dt}X^{\tau}(t) = H(t)X^{\tau}(t) \tag{0-29}$$

and it is evident that both $X(t)$ and $X^{\tau}(t)$ satisfy the same time-dependent Schrodinger equation. As such, we can surmize that the two are related through some constant and we will call that constant matrix R

$$X^{\tau}(t) = X(t)R \tag{0-30}$$