Solid-2.0 Documentation---

Solid-2.0 is a general quantum mechanical NMR simulation package designed to simulate virtually any pulse sequence NMR performs on up to 10 spins.

The distribution is with 'BlochLib' (because you need that in order to build solid), in the examples folder.

The source code interfaces directly with the BlochLib NMR library and is greatly simplified over previous versions, that addition of additional features is very simple...if you have any features that you would like to see (and have a knack for C++) read the BlochLib documentation first, then add away. If the feature is potent enough to warrant inclusion in the main distribution please let me know and i will add it (i'll even optimize it and extend it if possible)...if you are not so C++ savvy, then let me know the feature and i could add it...

Solid-2.0 is a major revision on the previous solid packages. It includes an in-house scripting capabilities, easier syntax, arbitrary 2D acquisitions and arbitrary point-to-point acquisitions, reusable pulse sections, multiple spin sections, manipulation of spin parameters in the pulse sequence. All in all, the entire package is much more powerful and useful. The old version is incompatible with this version.

This documentation is highly example driven with a brief description of each of the available function.

- **To Run In Parallel**
  - Make sure you compile with MPI
  - It runs on a master/slave model so when you run it make sure you add one extra processor (the 'master' only gives out work and does minimal work itself)

    If you have 4 procs..simply issue this command

    mpirun -np 5 solid {config file}

The 3 Sections

- Each input file should have these 3 sections with the basic syntax

```
spins{
    ...
}

parameters{
    powder{
    ...powder options...
    }

    ...other parameters...
}

pulses{
# if desired use a pulse section via
    sub1{
        ...pulses things....
    }

    ...main sequence running...
    ...and 'fid' collection...
}
```

comments can be made anywhere by beginning a line with '#'

- ## The Spins Section

You input file should have this section

```
spins{
...
}
```

Inside 'spins' you should obey the syntax as described for the input string vector of the 'SolidSys.' Spin indices are started at '0' (not 1) An example is given below

```
spins{
    numspins 2
#this defined the nucleous
#T {label} {spin}
    T 1H 0 #the first spin is a proton
    T 13C 1 #the second spin is 13C
    T 14N 2 # a quadrupole spin

#interactions are defined as below
# frequencies are defined in Hz (not
```

```
rad/sec)
#a CSA
#C {iso} {del} {eta} {spin} {alpha} {beta}
{gamma}
    C 3000 234 0.2 0 #a csa on the first
spin
    C 0 4567 0 1 #a csa on the second spin

#A Dipole
#D {del} {spin1} {spin2} {alpha} {beta}
{gamma}
    D 3450 0 1 # a dipole between spin 1 and
2

#a qudrupole
#Q {Q} {eta} {spin} {alpha} {beta} {gamma}
{order}
    Q 1e6 0 2 0 0 0 2

# scalar coupling
#J {J} {spin1} {spin2}
    J 500 0 1
}
```

The parameters Section

Your input file should have this section

```
parameters{
    powder{
    ...powder options...
    }

    ...other parameters...
}
```

You can define any variable you wish here using the syntax 'A=B' HOWEVER, there are some basic predefined variables that will be set regardless of weather or not they are set here. It should be noted that you can always alter these variables inside the 'pulses' section as well. These variables will be GLOBAL to any subpulse section and any item in the pulse section

The Global Variables

wr	the rotor spinning speed in Hz (default = 0)
rotor	the rotor angle in DEGREES (default =0)

maxtstep	the basic time step for performing the dyson time ordering for spinning samples. This value will be largest dt for any integration (default 1e-6 seconds)
npts1D	the number of FID points in the first dimension (default= 1)
npts2D	the number of FID points in the seond dimension (default = 1)
sw	The sweep width in Hz (default = 20kHz)
Bfield	The magnetic Field strength in Hz (default 400 Mhz)
roeq	the Initial Density matrix. It can be anything available to HamiltonianGen in BlochLib (default =Iz)
detect	The Detection matrix. It can be anything available to HamiltonianGen in BLochLib (default=Ip)
filesave	The file name to save the FIDs in (default=soliddata)

The powder subsection defines the powder average type, or the input file of angles. It follows the syntax available to the 'powder' class in BlochLib. Below is an example

```
# the internal zcw powder average
powder{
    aveType zcw
    thetaStep 233
    phiStep 144
    gammaStep 0
}

# using an input file
powder{
    aveType /total path to file/filename
}
```

The input file should have 2,3, or 4 columns and all angles are in RADIANS

```
2-column file:: {phi} {theta}
3-column file:: {phi} {theta} {weight}
4-column file:: {phi} {theta} {gamma} {weight}
```

Below is a typical parameter input example

```
parameters{
# the internal zcw powder average
    powder{
        aveType zcw
        thetaStep 233
        phiStep 144
```

```
            gammaStep 0
        }
        wr=3000
        rotor=acos(1/sqrt(3))*deg2rad #magic
    angle
        maxtstep=1e-6
        npts1D=512
        sw=10*wr
    }
```

- ## The pulses Section

  Your input file should have this section

```
    pulses{
    # if desired use a pulse section via
        sub1{
            ...pulses things....
        }

        ...main sequence running...
        ...and 'fid' collection...
    }
```

  This is the main driver. It performs the propagation, fid collection, and file saving. You can alter spin system parameters here, and change or define any variable you wish. Any variable you set here is GLOBAL to all the sub sections. The subsections should start with the 'sub1' and continue to 'subN.' Because the usage of this section is highly dependant on what you are doing, I'll simply move on to the function list, and let you look at the examples for more information.

The Pulse Section Function List....

ro(matrix)	sets the current density matrix to the input....thing can be of the syntax as HamiltonianGen in BlochLib
detect(matrix)	sets the detection matrix to the input....thing can be of the syntax as HamiltonianGen in BlochLib

amplitude(#)	Sets the default pulse amplitude (in Hz) the default will remain in effect until you call this function to set it again. The input can be any valid expression to Parser in BlochLib.
offset(#)	Sets the default pulse offset (in Hz) the default will remain in effect until you call this function to set it again. The input can be any valid expression to Parser in BlochLib.
ptop()	Sets the type of FID to a point to Point experiment. You must set this ANYTIME you want to use the fid() command to collect only ONE point. It performs a trace of the current evolved 'ro' with the detection matrix. You must declare this BEFORE the 'fid' command is used.
2D()	If you are collecting ANY TYPE of 2D data, you must set this flag so the data get handled properly. You must declare this BEFORE the 'fid' command is used.
use(subsect)	This tells the program to use a subsection defined as a 'sub1'..'subN' section. It will RECALCULATE it propagator every time it sees this function. You should use this function (as aposed to 'reuse') if there is a variable inside the subsection that gets updated OUTSIDE the subsection which would cause the previously calculated propagator to be invalid (things like changing the rotor angle, or a pulse amplitude)
reuse(subsect)	This tells the program to use a subsection defined as a 'sub1'..'subN' section. It will CALCULATE the propagator ONLY ONCE. You should use this function when the propagator will remain the same regardless of the time, or any other variables. For example a C7 uses a single 2 rotor cycle that does not change the propagator for each time incremented. This saved you valuable computation time when used properly.

use(subsect, repeat use(subsect, repeat, hold)	Performs the same things as the use or reuse command above, except that if the subsection is a time periodic, then you can save computational time by simply 'repeating' the propagator rather then recalculating the entire thing. The parameter 'repeat' should evaluate to a number >=1.
reuse(subsect, repeat) reuse(subsect, repeat, hold)	The hold command is useful when doing 2D type experiments (either a ptop or a full 2D collection). The propagator will only be applied at the FIRST point in any fid. Things like cross-polarization (CP) fit into this category where one only CP's at the beginning of the sequence.
cycler(matrix)	This command enables you to do 'fictitious' phase cycling. Whenever this command is implemented the current propagated density matrix will be 'traced' with the input matrix such that the resulting density matrix becomes ro=trace(adjoint(cycler), ro)*cycler For example a z-filter (used commonly to remove unwanted X and Y coherences would be implemented by 'cycler(Iz).' The matrix can be of the syntax as HamiltonianGen in BlochLib.
fid()	Collects an ENTIRE fid of the length of the 'npts1D' variable, weather that be a point to point experiment, static FIDs, or spinning FIDs.
fid(int)	This is an extension that give you the ability to specify what fid point you are collecting. To use this for 1D FIDs you must have set the ptop flag using 'ptop().' To use this for 2D fids you must use the '2D()' flag. For Point-To-Point 1D experiments a single complex value is added to the data vector, for 2D data an FID of length npts1D is computed (either in the normal static, spinning approach or the Point To Point approach (if the ptop() command was used)).

alterSys(what, num)	This allows you to alter a parameter in the SpinSystem. The 'what' is of the syntax D01del --> alters the dipolar coupling between 0 and 1 C1iso --> alters the isotropic shift of spin 1 C0eta --> alters the eta of spin 0 Q0del --> alters the quad coupling on spin 1 D01alpha --> changes the orientation angle of the dipole between 0 and 1 The couplings MUST be defined in the 'spins' section before you can alter them...num can be of any expression valid to the Parser class in BlochLib
savefid() savefid(name) savefid(name, which)	This will save the fid AFTER the ENTIRE powder average has been calculated. It will automatically determine the save type. If the spectra is 1D it will save it as a 'text' file (see below). If the data is 2D it will save it as a matlab binary file (see below). If 'name' is not present it will use the variable 'filesave' as the file name. If 'which' is present it will save a 2D fid in its 1D components adding the number 'num' to the end of the file name to distinguish it from the rest...this is only valid for 2D data.
savefidtext() savefidtext(name) savefidtext(name, which)	Does the same as the above function except forces it to save the data as an ASCII file (for both 1 and 2Ds)
savefidmatlab() savefidmatlab(name) savefidmatlab(name, which)	Does the same as the above function except forces it to save the data as a matlab file (for both 1 and 2Ds)
savefidbinary() savefidbinary(name) savefidbinary(name, which)	Does the same as the above function except forces it to save the data as a binary file (for both 1 and 2Ds)
show()	This command DISPLAYS what the simulator 'would be doing' if you do not call this function. It dumps a LARGE variety of information to the screen and can be useful for debugging you pulse sequences. You should set this at the beginning of the pulses section such that everything is displayed. It does NOT calculate anything (except the variables and inputs)

{spin}:pulse(time, phase, amp, offset)	This produces a pulse on spin {spin} for the time {time} in seconds, amplitude {amp} in Hz, phase {phase} in DEGREES, and offset {offset} in Hz...some examples a 90 pulse on proton 1H:pulse(1/15000/4, 90, 15000, 0) uses the default offset (by 'offset(#)') 1H:pulse(1/15000/4, 90, 15000) uses the default amplitude (by 'amplitude(#)') and offset 1H:pulse(1/15000/4, 90) To perform multiple pulse on different spins simply use the '|' a 90 pulse on both carbon and 1H amplitude(15000) 1H:pulse(1/15000/4,90) | 13C:pulse(1/15000/4,90) if there are more then one spin type in the system, and no pulse is specified for it at that time, a DELAY is assumed...
{spin}:delay(time)	This produces a delay (simple evolution under the current Hamiltonian) for a time {time} in seconds Some examples 1H:delay(0.001) 1H:pulse(1/15000/4,90) | 13C:delay(1/15000/4)

File Formats...

- **Text**

  For all 1D data, the output file will contain these columns

  {time} {frequency} {real fid} {imag fid} {real fft} {imag fft} {power fft}

For all 2D data, the file will contain

```
npts1 = {npts1D}
npts2 = {npts2D}
{matrix row index} {matrix col index} {real part} {imag part}
...
```

- **Binary**

  For all 1D data, the ouput will be

  ```
  npts1={npts1D}
  {real val 0}{imag val 0}...{real val N}{imag val N}
  ```

  the 'real val ' and 'imag val' are in binary

  For all 2D data, the output will be

  ```
  npts1={npts1D}
  npts2={npts2D}
  {real val row=0, col=0}{imag val row=0, col=0}...
  {real val row0, col=npts2}{imag val row=0, col=npts2}
  ...
  {real val row=npts1, col=0}{imag val row=npts1, col=0}...
  {real val row=npts1, col=npts2}{imag val row=npts1,
  col=npts2}
  ```

  the 'real val ' and 'imag val' are in binary

- **Matlab**

  For both 1D and 2D data, a data matrix (or vector) will be saved in matlab format with the name "vdat" (so that it interfaced directly with 'solidplotter')

Contact: Bo Blanton

Solid-2.0 Documentation---Static Examples

Below are several examples of performing static simulations (things where the spinning speed it set to 0). You could easily make these into spinning by setting the wr!=0.

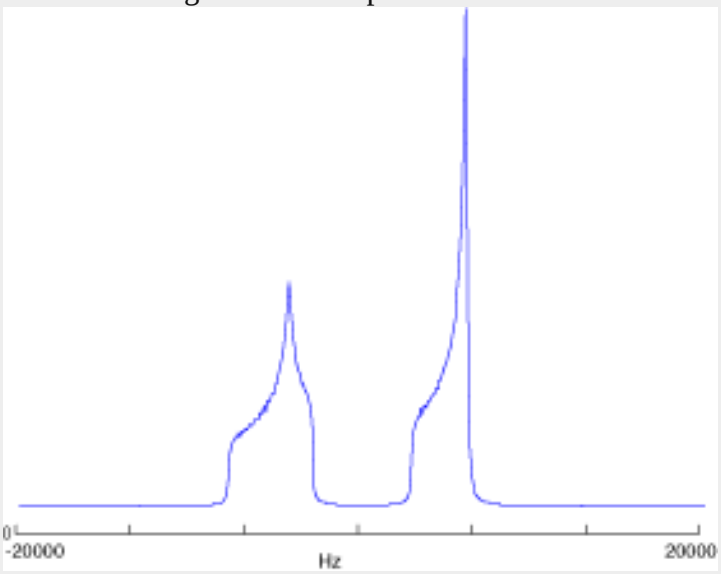
Basic

- Below is an input file the a basic static simulation of 2 CSAs...the crystal file came from the BlochLib distribution.

```
spins{

     #the global options
     numspin 2
     T 1H 0
     T 1H 1

     C 5000 2134 0 0
     C -5000 2789 0.5 1
}

parameters{

     powder{
          aveType ../../../../crystals/rep2000
     }

#the intergrator step size
     maxtstep=5e-6

#number of 1D fid points
     npts1D=512

#sweepwidth
     sw=40000

     roeq= Iz
     detect=Ip
     filesave=data
}


pulses{

#set the spinning
     wr=0
#set the rotor
     rotor=0
#set the detection matrix
     detect(Ip)
#set the inital matrix
```

```
    ro(Ix)

#no pulses nessesary for ro=Ix

#collect the fid
    fid()
    savefidtext(simpSTA) #save as a text file
}
```

Here is the generated spectrum



Basic 2D using multi sections

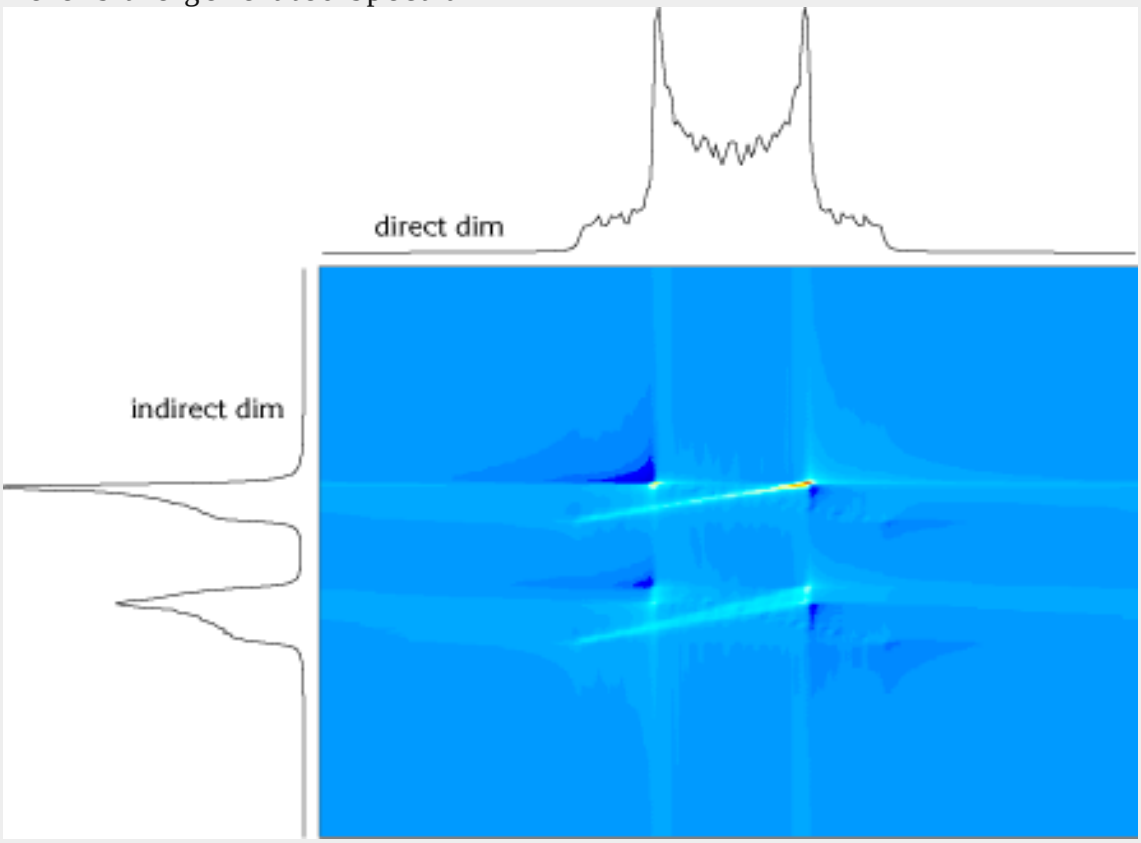
- Below is an input file the a basic static simulation of '2D' in the direct dimension we use the Dipole ONLY spin system, and in the indirect dimension we use the CS spin system....

```
# a simple static spectra using 2 differnet spin
# systems for each dimension
# using the basic algorithms

spins{

   #the global options
   numspin 2
   T 1H 0
   T 1H 1

   spin1{
      C 5000 2134 0 0
      C -5000 2789 0.5 1
   }

   spin2{
      D 1254 0 1
   }
```

```
}


parameters{

   powder{
      aveType ../../../../crystals/rep678
   }

#the intergrator step size
   maxtstep=1e-6

#number of 1D fid points
   npts1D=256

#sweepwidth
   sw=10000

#the eq matrix
   roeq=Ix
}


pulses{

#our 2D points
   fidpts=256
   2D()
#set the spinning
   wr=0
#set the rotor
   rotor=0

#set the detection matrix
   detect(Ip)
#set our initial matrix
   ro(Ix)
   dwell2D=0.00002
#set the inital matrix
   loop(i=0:fidpts-1)

   #use the second spin system for the direct dim
      spinsys(spin2)

   #collect the fid (to get the first point)
      fid(i)

   #do not need to propogate the last point
      if(i!=(fidpts-1))
      # 'indirect dim' spin system
         spinsys(spin1)

      #a delay for the second dim
         1H:delay(dwell2D)
      end
```

```
    end
    savefidmatlab(2dstat) #save as a matlab file
}
```

Here is the generated spectra



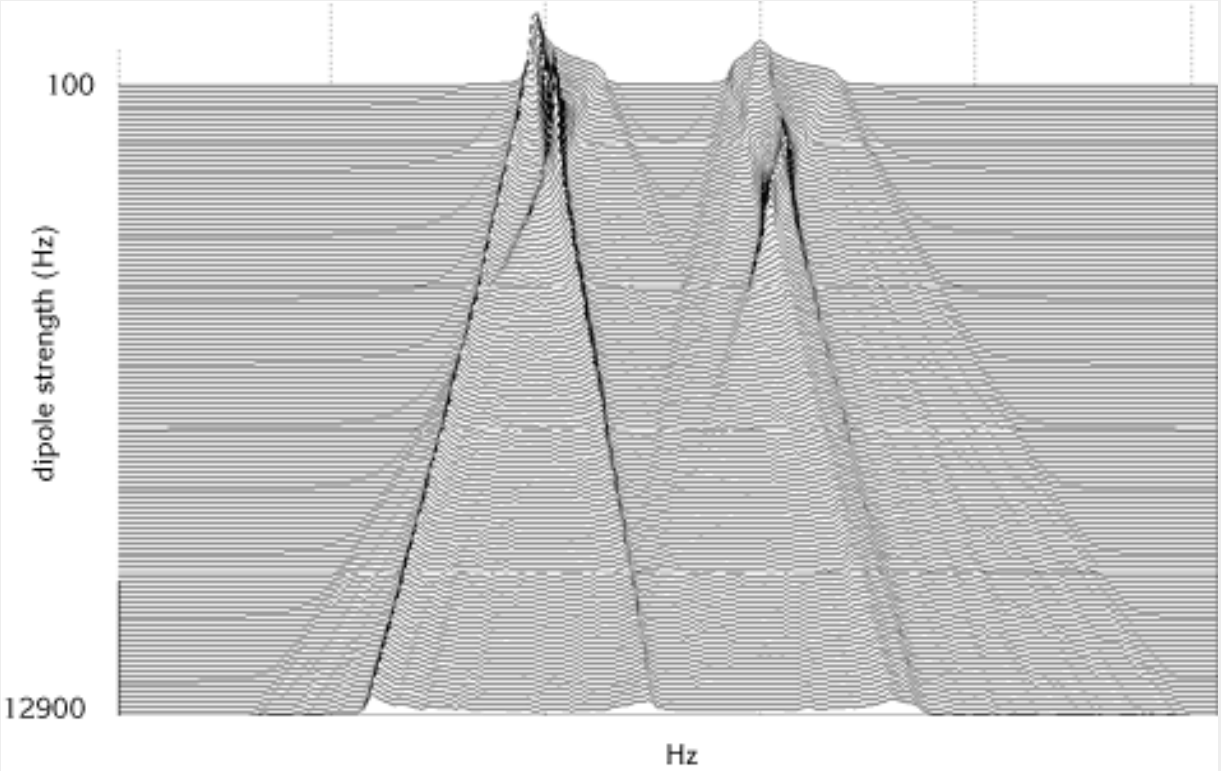
a series of 1Ds (a 2D data set)

- An example demonstrating how to use 'alterSys' to collect various fids for different dipolar couplings on the system

```
# a simple static spectra using that loops
# through a bunch of dipole couplings and places them in a 2D set

spins{
    #the global options
    numspin 2
    T 1H 0
    T 1H 1
    C 5000 2134 0 0
    C -5000 2789 0.5 1
    D 1254 0 1

}


parameters{
```

```
    powder1{
      aveType ../../../../crystals/rep678
      thetaStep 233
      phiStep 144
      gammaStep 0
    }

  #the intergrator step size
    maxtstep=1e-6

  #number of 1D fid points
    npts1D=256

  #sweepwidth
    sw=50000

  #the eq matrix
    roeq=Ix
  }


  pulses{

  #our 2D points
    fidpts=128
    2D()

  #set the spinning
    wr=0

  #set the rotor

    rotor=0
  #the dipole step
    dipStep=100
    dip=100

  #set the detection matrix
    detect(Ip)

    loop(i=0:fidpts-1)
      ro(Ix) #reset the desity matrix

    #change the dipole copling
      alterSys(D01, dip)
      dip=dip+dipStep

    #collect the fid
      fid(i)
    end
    savefidmatlab(2dalter) #save as a matlab file
  }
```

Here are the generated spectra



a 1D decoupling experiment over the the dcoupling amplitude

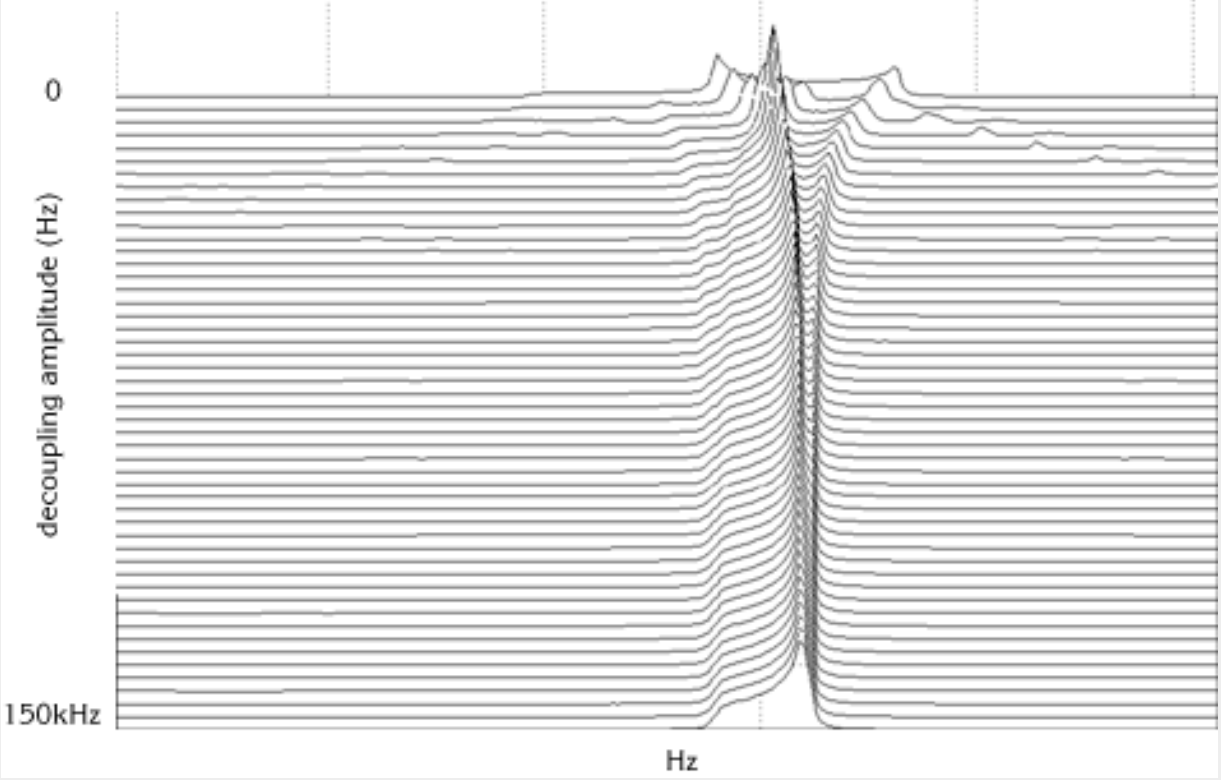
- An example demonstrating how to use the fid(i) to collect a 1D fid that demonstrates CW-decoupling between a 1H and 13C. Below are a series of 1D point-to-point experiments that loop through a series of dceoupling amplitudes.

```
# A static decoupling sim
# this is by nessescity a point to point
# as there is a pulse during the fid collection

spins{

   #the global options
   numspin 2
   T 1H 0
   T 13C 1
   C 5000 2134 0.8 0
   C 5000 2789 0 1
   D 8300 0 1

}


parameters{

   powder1{
      aveType zcw
      thetaStep 377
      phiStep 233
      gammaStep 0
```

```
   }


#number of 1D fid points
   npts1D=256

}


pulses{


#a point to point
   ptop()
   2D()
#the number of decouple amplitudes to use
   dcpts=50

#set the spinning
   wr=0

#set the rotor
   rotor=0

#set the detection matrix
# detect the 13C
   detect(Ip_1)

#the decouple amplitude
   dcamp=0
   dcstep=150000/dcpts

#our dwell
   dwell=1/50000

#NOTE: this would be invalid for
# wr>0 as the calulation would require
# a time dependant propogaor!
   sub1{
      1H:pulse(dwell, 0, dcamp) | 13C:delay(dwell)
   }


   loop(i=0:dcpts-1)
   #set the desity matrix to a pulse 13C and
   # not pulsed 1H
      ro(Ix_1+Iz_0)

   #must 'use' the subsection as
   # the dcamp changes
      use(sub1)

   #collect the fid
      fid(i)
```

```
    #advance the dc amplitude
      dcamp=dcamp+dcstep

    end
    savefidmatlab(decoup) #save as a matlab file
}
```

Here are the generated spectra



Contact: Bo Blanton

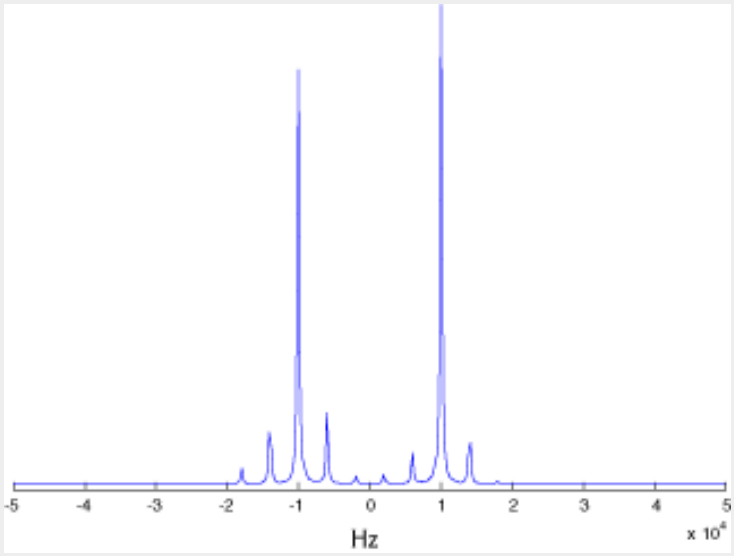
Below are several examples of performing spinning simulations.

Basic

- Below is an input file the a basic MAS simulation of 2 CSAs...the crystal file came from the BlochLib distribution.

```
spins{

     #the global options
     numspin 2
     T 1H 0
     T 1H 1

     C 5000 2134 0 0
     C -5000 2789 0.5 1
}

parameters{

     powder{
          aveType ../../../../crystals/rep256
     }

#the intergrator step size
     maxtstep=1e-6

#number of 1D fid points
     npts1D=512

#sweepwidth
     sw=40000

     roeq= Iz
     detect=Ip
     filesave=data
}


pulses{

#set the spinning
     wr=2000
#set the rotor to the magic angle
     rotor=rad2deg*acos(1/sqrt(3))
#set the detection matrix
     detect(Ip)
#set the inital matrix
     ro(Ix)

#no pulses nessesary for ro=Ix

#collect the fid
     fid()
     savefidtext(simpMAS) #save as a text file
}
```

Here is the generated spectrum



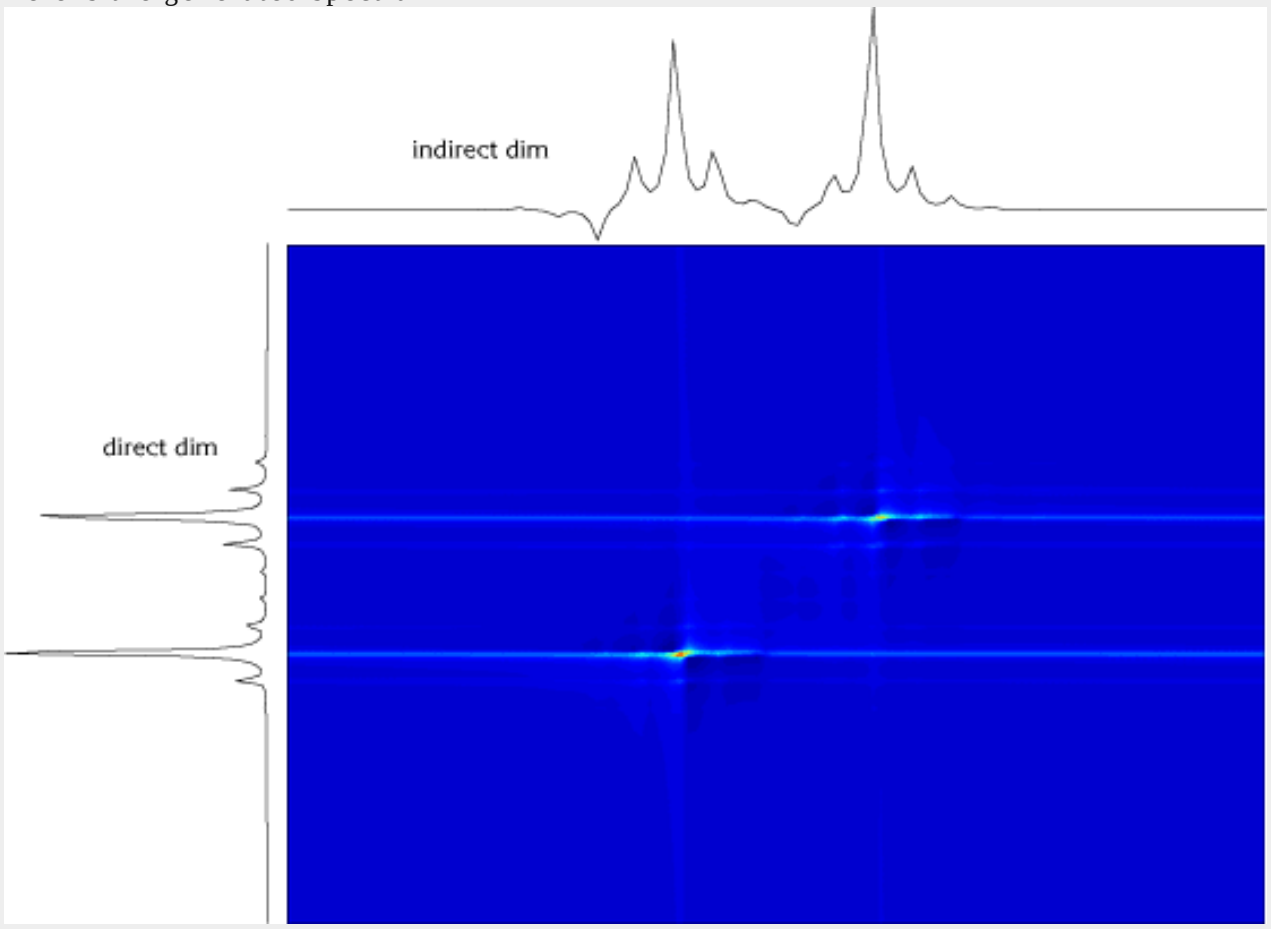
Basic 2D

- Below is an input file the a basic static simulation of '2D' both projection should give exactly the same thing...a boring example, but a proof of point.

```
spins{

   #the global options
  numspin 2
  T 1H 0
  T 1H 1

  C 5000 2134 0 0
  C -5000 2789 0.5 1
}


parameters{

   powder{
      aveType ../../../../crystals/rep256
   }

#the intergrator step size
   maxtstep=1e-6

#number of 1D fid points
   npts1D=256

#sweepwidth
   sw=10000

#the eq matrix
   roeq=Ix
}


pulses{

#our 2D points
   fidpts=256
   2D()
```

```
#set the spinning
   wr=2000
#set the rotor
   rotor=rad2deg*acos(1/sqrt(3))

#set the detection matrix
   detect(Ip)
#set our initial matrix
   ro(Ix)
   dwell2D=0.00002
#set the inital matrix
   loop(i=0:fidpts-1)
   #collect the fid (to get the first point)
      fid(i)

   #do not need to propogate the last point
      if(i!=(fidpts-1))
      #a delay for the second dim
         1H:delay(dwell2D)
      end
   end
   savefidmatlab(2dmas) #save as a matlab file
}
```

Here is the generated spectra



MAS the old fashioned way

- This example collects an MAS fid using the 'direct' method. With methods advances the propogator in a point-by-point fashion. This is much slower then the compute method usually used, but this just goes to show that you can do it if you wish.

```
# a simple MAS collection
# using the basic algorithms
# this one is essentially the same as perfomring a 'direct'
# computation of the fid (dyson time series)

spins{
   #the global options
   numspin 2
   T 1H 0
   T 1H 1
   C 5000 2134 0 0
   C -5000 2789 0.5 1
}

parameters{

   powder1{
      aveType ../../../../crystals/rep256
   }

#the intergrator step size
   maxtstep=1e-6

#number of 1D fid points
   npts1D=256
}


pulses{
   ptop()
#set the spinning
   wr=3000
#set the rotor
   rotor=rad2deg*acos(1/sqrt(3))
#set the detection matrix
   detect(Ip)
#set our initial matrix
   ro(Iz)

   dwell2D=0.00002

#give our spins a 90
   1H:pulse(1/150000/4, 0, 150000)

   loop(i=0:npts1D-1)
   #collect the fid
      fid(i)

      if(i!=(npts1D-1))
         1H:delay(dwell2D)
      end
   end

   savefidtext(ptopMAS) #save as a text file
```

}

The generated spectra will look like the first 'basic' example, however, i have noticed several 'phase-noise' that become present when performing the 'driect' dyson series for spinning simulations. There are thousands of propogators that need to be calculated and if the time dt step is choosen to be too small it will devolope both frequecy and phase errors. These errors will propogate thus creating undesired peaks and oddly phased peaks.

Contact: Bo Blanton

Below are several examples of performing the good old recoupling sequence 'post-C7.' This sequence is a fine example demonstrating all the various techniqes and pulse sequences that exsist that are rotor syncronized.

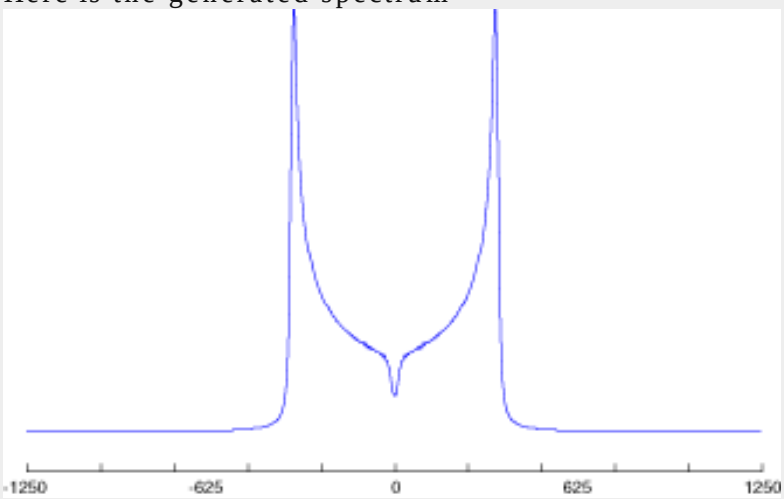
Basic

- Below is an input file the a basic MAS simulation of 2 CSAs...the crystal file came from the BlochLib distribution.

```
# preforms a simple point-to-point C7 (a 1D FID)

spins{

   #the global options
   numspin 2
   T 1H 0
   T 1H 1
   D 1500 0 1
}

parameters{

   powder{
      aveType zcw
      thetaStep 233
      phiStep 144
   }

#the intergrator step size
   maxtstep=1e-6

#number of 1D fid points
   npts1D=512

   roeq= Iz
   detect=Ip
   filesave=data
}


pulses{

#our post-C7 sub pulse section
   sub1{
   #set the rotor angle to the global var
      rotor=rad2deg*acos(1/sqrt(3))
   #post C7 pulse amplitude
      amp=7*wr
      amplitude(amp)
   #phase stepers
      stph=0
      phst=360/7
   #pulse times
      t90=1/amp/4
      t270=3/amp/4
      t360=1/amp
```

```
   #post C7 loop
      loop(k=1:7)
         1H:pulse(t90, stph)
         1H:pulse(t360, stph+180)
         1H:pulse(t270, stph)
         stph=stph+phst
      end
   }

#a single fid is concidered point to point
   ptop()

#set the spinning
   wr=5000
   rotor=rad2deg*acos(1/sqrt(3))

#set the detection matrix
   detect(Iz)

#can use 'reuse' as things parameters are set once
# in our subsection
   reuse(sub1)

#collect the fid
   fid()
   savefidtext(simpC7) #save as a text file
}
```

Here is the generated spectrum



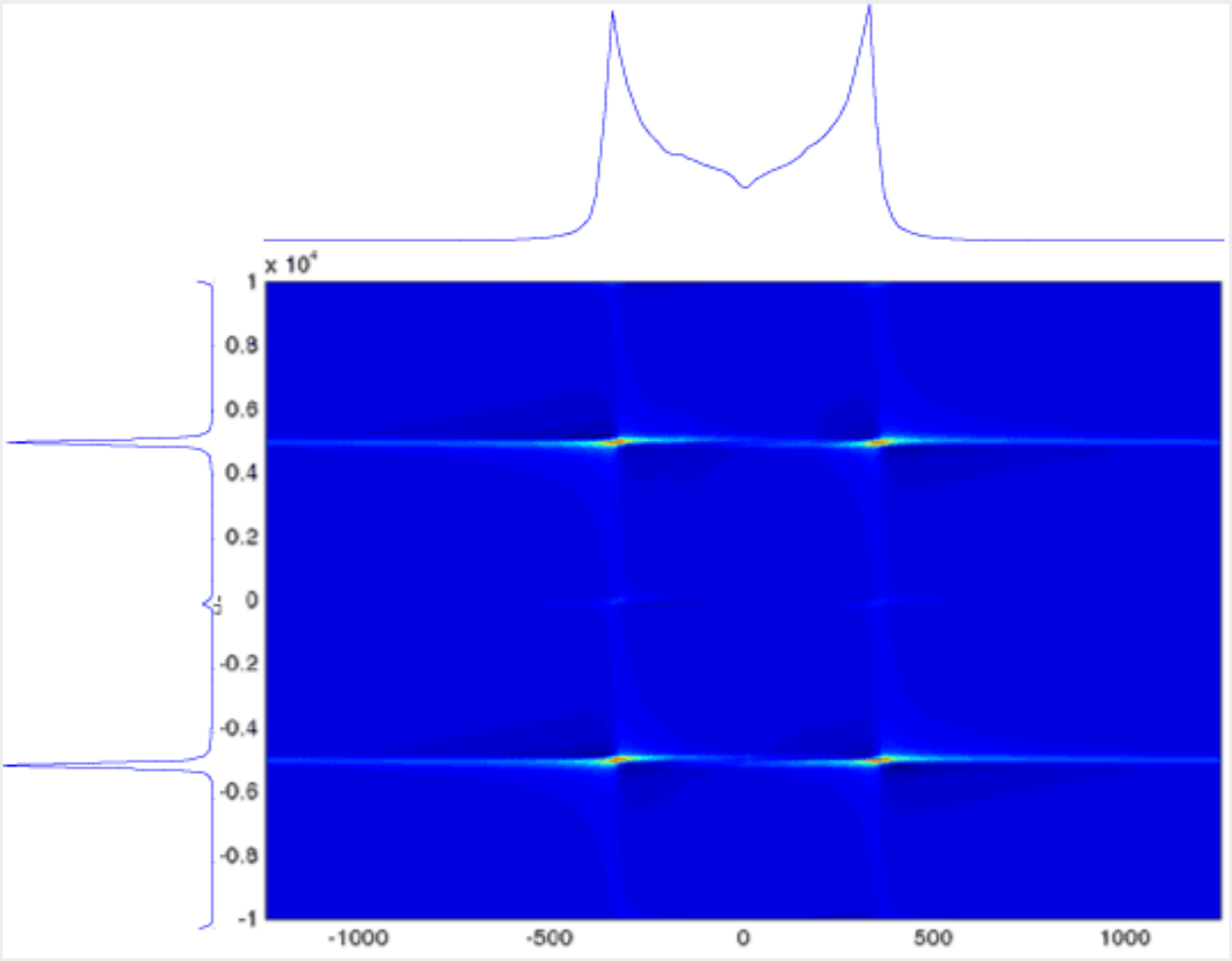
The real 2D experiment

- In reality it is hard to collect 'one-point' of an fid and have everything function properly, soa basic post-C7 experiment is performed in a 2D fashion

```
# preforms a 'real' experiment
# for the post-C7 (a series of 2D fids are collected)

spins{
   #the global options
   numspin 2
   T 1H 0
   T 1H 1
   C 5000 2134 0 0
   C -5000 2789 0.5 1
   D 1500 0 1
}

parameters{

   powder{
      aveType zcw
      thetaStep 233
      phiStep 144
   }

#the intergrator step size
   maxtstep=1e-6

#number of 1D fid points
   npts1D=512
}


pulses{

#our post-C7 sub pulse section
   sub1{
   #set the rotor angle to the global var
      rotor=rad2deg*acos(1/sqrt(3)))
   #post C7 pulse amplitude
      amp=7*wr
      amplitude(amp)
   #phase stepers
      stph=0
      phst=360/7
   #pulse times
      t90=1/amp/4
      t270=3/amp/4
      t360=1/amp

   #post C7 loop
      loop(k=1:7)
         1H:pulse(t90, stph)
         1H:pulse(t360, stph+180)
         1H:pulse(t270, stph)
         stph=stph+phst
      end
   }

#number of 2D points
```

```
   fidpt=128

#collection a matrix of data
   2D()

#set the spinning
   wr=5000

#the basic rotor angle
   rotor=rad2deg*acos(1/sqrt(3)))

#set the detection matrix
   detect(Ip)
#reset the ro back to the eq
   ro(Iz)

#90 time ampltiudes
   amp=150000
   t90=1/amp/4

#loop over the fids
   loop(m=0:fidpt-1)

   #may use 'reuse' as everything is static in sub1
   # must be repeat m times to advance the desity matrix
   # for each fdi (the first fid gets no c7)
      reuse(sub1, m)

   #pulse the IZ down to the xy plane for detection
      1H:pulse(t90, 270, amp)
  #collect the fid at the 'mth' position
      fid(m)

   #reset the ro back to the eq
      ro(Iz)
   end
   savefidmatlab(2dc7) #save the matlab file
}
```

Here is the generated spectra



---

doing a post-C7 explicitly

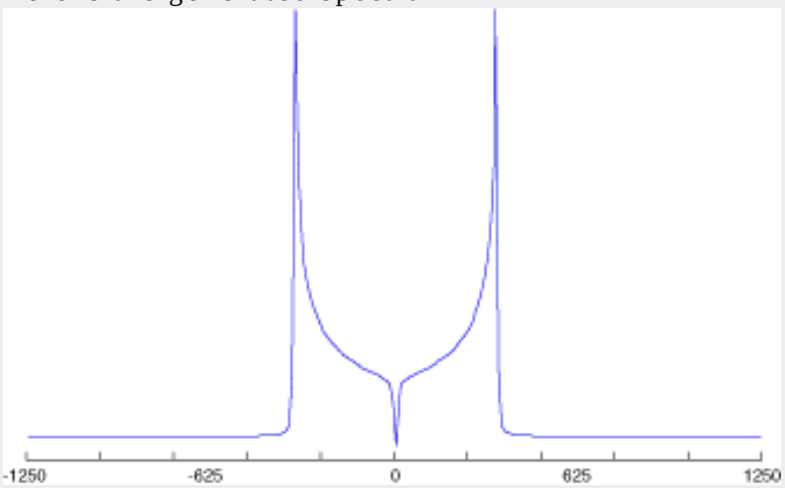
- This performs the point-to-point by hand rather then using the internal loop for the ptop...it will be slower then the 'basic' one above

```
# preforms a simple point-to-point C7 (a 1D FID)
# put rather letting the program do the ptop
# it does the point collection explicitly
# this will be slower then simply setting the ptop
# flag becuase it relies on non-compile code
# to do the loop....

spins{

   #the global options
   numspin 2
   T 1H 0
   T 1H 1
   D 1500 0 1
}

parameters{

#our basic powder average
   powder{
      aveType zcw
```

```
      thetaStep 233
      phiStep 144
   }

#the intergrator step size
   maxtstep=1e-6

#number of 1D fid points
   npts1D=256

}


pulses{

#our post-C7 sub pulse section
   sub1{
   #set the rotor angle
      rotor=rad2deg*acos(1/sqrt(3))
   #post C7 pulse amplitude
      amp=7*wr
      amplitude(amp)
   #phase stepers
      stph=0
      phst=360/7
   #pulse times
      t90=1/amp/4
      t270=3/amp/4
      t360=1/amp

   #post C7 loop
      loop(k=1:7)
         1H:pulse(t90, stph)
         1H:pulse(t360, stph+180)
         1H:pulse(t270, stph)
         stph=stph+phst
      end
   }

#a single fid is concidered point to point
   ptop()

#set the spinning
   wr=5000

#set the detection matrix
   detect(Iz)

#set the density matrix
   ro(Iz)
   loop(i=0:npts1D-1)
   #collect the fid at point i
      fid(i)

   #can use 'reuse' as things need to be set only once
   # in our subsection
      reuse(sub1)

   # do NOT set ro back to equilibrium as
   # we want the last ro to be used for the next point
```

```
    end
    savefidtext(simpC7ex) #save as a text file
}
```

Here is the generated spectrum



---

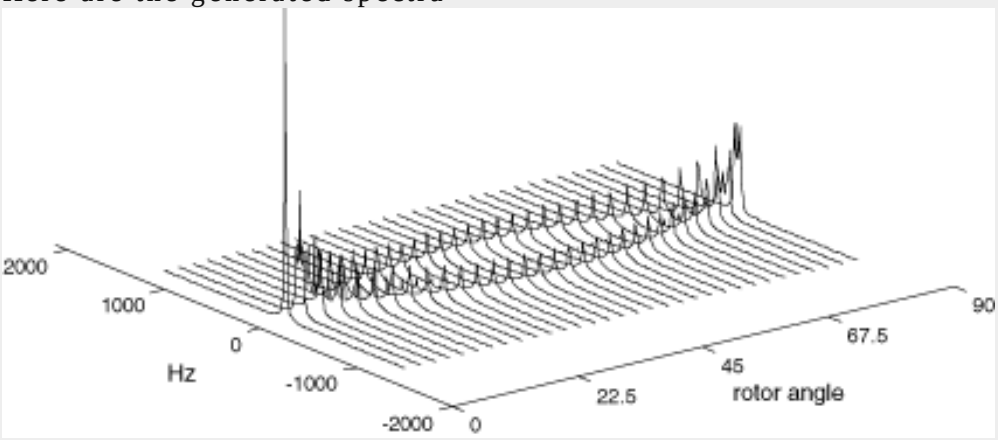
post-C7 dependance on rotor angles...

- this simply collects a bunch of post-C7s at different rotor angles

```
#the multplit spin list
# preforms a simple post-C7 over
# different rotor angles

spins{

   #the global options
   numspin 2
   T 1H 0
   T 1H 1
   D 1500 0 1
}

parameters{

   powder{
      aveType zcw
      thetaStep 233
      phiStep 144
   }

#the intergrator step size
   maxtstep=1e-6

#number of 1D fid points
   npts1D=512
}


pulses{
```

```
#our post-C7 sub pulse section
   sub1{
   #set the rotor angle to the global var
      rotor=myR
   #post C7 pulse amplitude
      amp=7*wr
      amplitude(amp)
   #phase stepers
      stph=0
      phst=360/7
   #pulse times
      t90=1/amp/4
      t270=3/amp/4
      t360=1/amp

   #post C7 loop
      loop(k=1:7)
         1H:pulse(t90, stph)
         1H:pulse(t360, stph+180)
         1H:pulse(t270, stph)
         stph=stph+phst
      end
   }

   fidpt=32

#collection a matrix of data
   2D()

# concidered point to point
   ptop()

#set the spinning
   wr=5000
# set the rotor angle steps
   rotst=90/fidpt
#the basic rotor angle
   myR=0

#set the detection matrix
   detect(Iz)

#loop over the rotor steps
   loop(m=0:fidpt-1)

   #must use 'use' as the rotor angle changes
      use(sub1)
   #collect the fid at the 'mth' position
      fid(m)

   #advance the rotor angle
      myR=myR+rotst
   #reset the ro back to the eq
      ro(Iz)
   end
   savefidmatlab(c7rotor) #save the matlab file
}
```

Here are the generated spectra



---

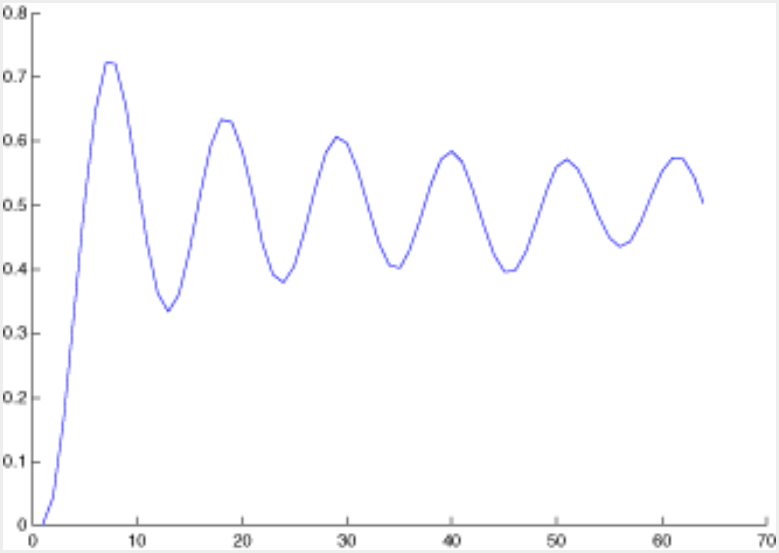
post-C7 coherence transfer

- post-C7 is also know for its ability to tranfer coherences between spins. To show this transfer, we simply detect the amount of signal generated on the oposing spin starting when we started with none.

```
#the multplit spin list
# preforms a simple point-to-point C7 (a 1D FID)
# and observers the coherence transfer
# between the two spins...

spins{

   #the global options
   numspin 2
   T 1H 0
   T 13C 1
   D 1500 0 1
}

parameters{

   powder{
      aveType ../../../../crystals/rep678
   }

#the intergrator step size
   maxtstep=5e-6

#number of 1D fid points
   npts1D=256
   roeq=Iz
}


pulses{

#our post-C7 sub pulse section
   sub1{
   #set the rotor angle to the global var
      rotor=rad2deg*acos(1/sqrt(3))
```

```
    #post C7 pulse amplitude
      amp=7*wr
      amplitude(amp)
    #phase stepers
      stph=0
      phst=360/7
    #pulse times
      t90=1/amp/4
      t270=3/amp/4
      t360=1/amp

    #post C7 loop
      loop(k=1:7)
        1H:pulse(t90, stph) | 13C:pulse(t90, stph)
        1H:pulse(t360, stph+180) |13C:pulse(t360, stph+180)
        1H:pulse(t270, stph)| 13C:pulse(t270, stph)
        stph=stph+phst
      end
    }


  #a single fid is concidered point to point
    ptop()

  #set the spinning
    wr=5000
    rotor=rad2deg*acos(1/sqrt(3))

  #set the ro to all Iz_1 and no Iz_0
    ro(Iz_0)

  #set the detection matrix
  # just detect the first spins
  # increase in its coherence
    detect(-Iz_1)

  #can use 'reuse' as things parameters are set once
  # in our subsection
    reuse(sub1)

  #collect the fid
    fid()
    savefidtext(transC7) #save tas a text file
  }
```

Here is the generated transfer



Contact: Bo Blanton

Solid-2.0 Documentation---Quadrupole Examples

Below are several examples of performing simulations on qudrupolars.

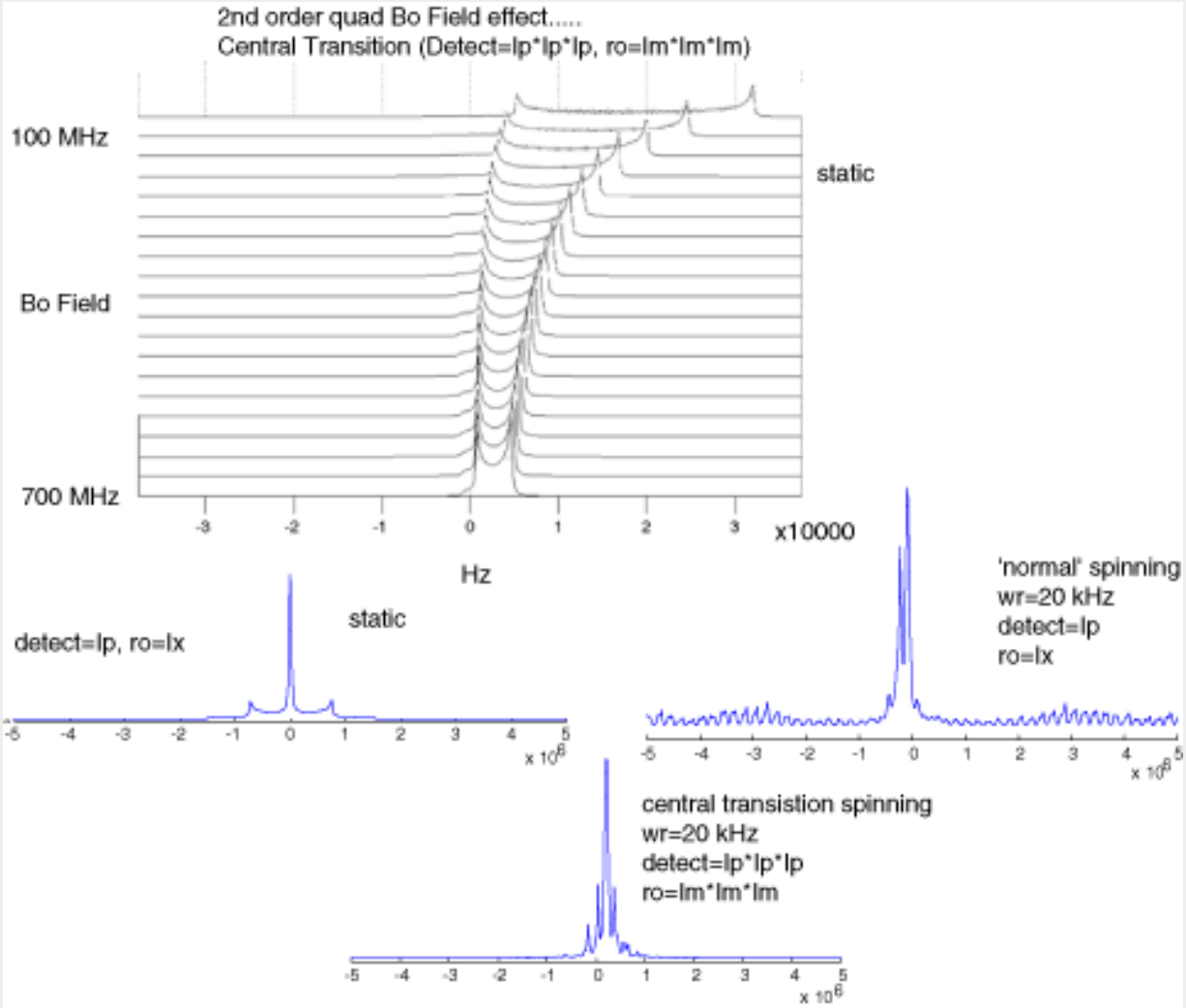
Central Transistion

- Becuase quadrupoles have very large coupling and are > spin 1/2, they represent a different type of NMR system. There is both a first order effect and a second order effect. The second order comes around becuase the couplings are so large (comperable to the magnetic field). The central transistion for quadrupoles is devoid of any anisontropy due to the first order, however, it is still effected by the second order. Below is the basic input file for the below figures. Simply changing the detection and initial density matrices will produce the below spectra. The second order effect is field dependant, and a loop over the Bfield variable will produce the 2D plot below as well.

```
# a basic quadrupole central transistion observation...you can
# simply change the spinning speed, detect, and ro to get the desired
# spectra shown below

spins{
   #the global options
   numspin 1
   T 23Na 0
   Q 3e6 0 0
}

parameters{

   powder{
       aveType zcw
      thetaStep 377
      phiStep 233
   }

#the intergrator step size
   maxtstep=1e-6

#number of 1D fid points
   npts1D=512

#sweepwidth
   sw=1000000
#the magnetic field
   Bfield=400e6
}


pulses{

# a 2D to loop over field strengths
   2D()

   BFpts=20
```

```
   BFstart=100e6
   BFend=700e6
   BFsteps=(BFend-BFstart)/BFpts

#set the spinning
   wr=0
#set the rotor
   rotor=rad2deg*acos(1/sqrt(3))
#set the detection matrix

# the central transistion 'top' (+1)
   detect(Ip*Ip*Ip)

#loop of over the field strengths
   loop(i=0:BFpts-1)
      Bfield=BFstart

   #set the inital matrix
   # the central transistion 'bottom' (-1)
      ro(Im*Im*Im)

#no pulses nessesary

   #collect the fid
      fid(i)
      BFstart=BFstart+BFsteps
   end
   savefidmatlab(bofields) #save as a matlab file
}
```

Here is the generated spectra, simply changeing the detection, ro, and the spinning speeds from the above input file.



 Contact: Bo Blanton