

Quantum for Mantid “table driven” instructions

What is Quantum?

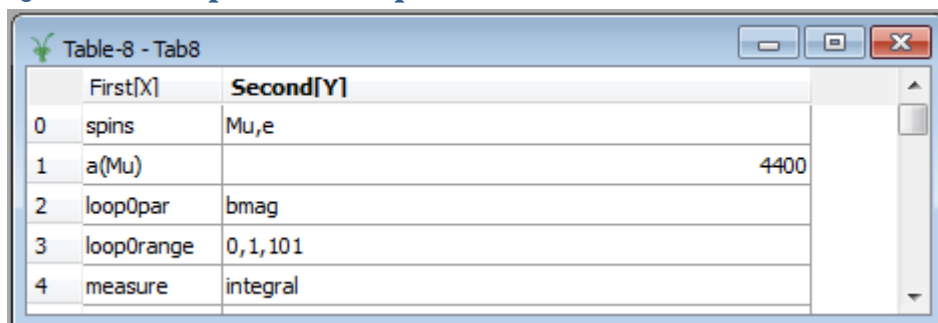
A program to solve the time evolution of the muon spin, given various interactions such as dipolar, hyperfine or quadrupole splitting, and applied magnetic fields. Using fitting, it can work back from experimental data to find the unknown interactions. It uses the density matrix method.

What doesn't it do?

Quantum only works with spin degrees of freedom, not spatial wavefunctions. If you want to calculate the hyperfine coupling of a radical from first principles you will need to use a DFT code such as “Gaussian” – but you can then use its hyperfine constants in Quantum to calculate the level crossing spectrum.

Quantum works with a finite number of coupled spins. If you're modelling a ferromagnet with stronger coupling among the spins in the lattice than to the muon, and long range dipolar fields, a classical lattice sum may be more appropriate.

Quick example for a repolarisation curve:



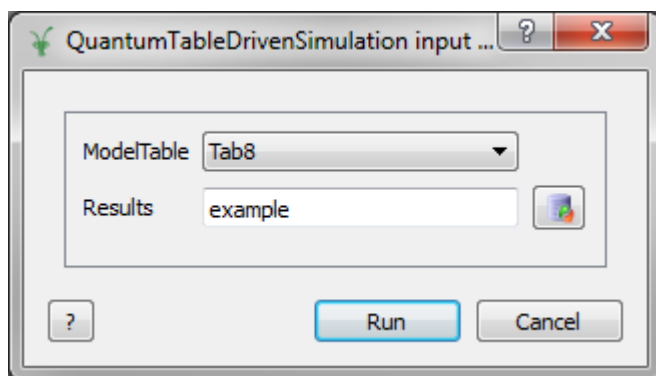
	First[X]	Second[Y]
0	spins	Mu,e
1	a(Mu)	4400
2	loop0par	bmag
3	loop0range	0,1,101
4	measure	integral

Line 0 defines the spins involved, and implicitly sets the gyromagnetic ratios.

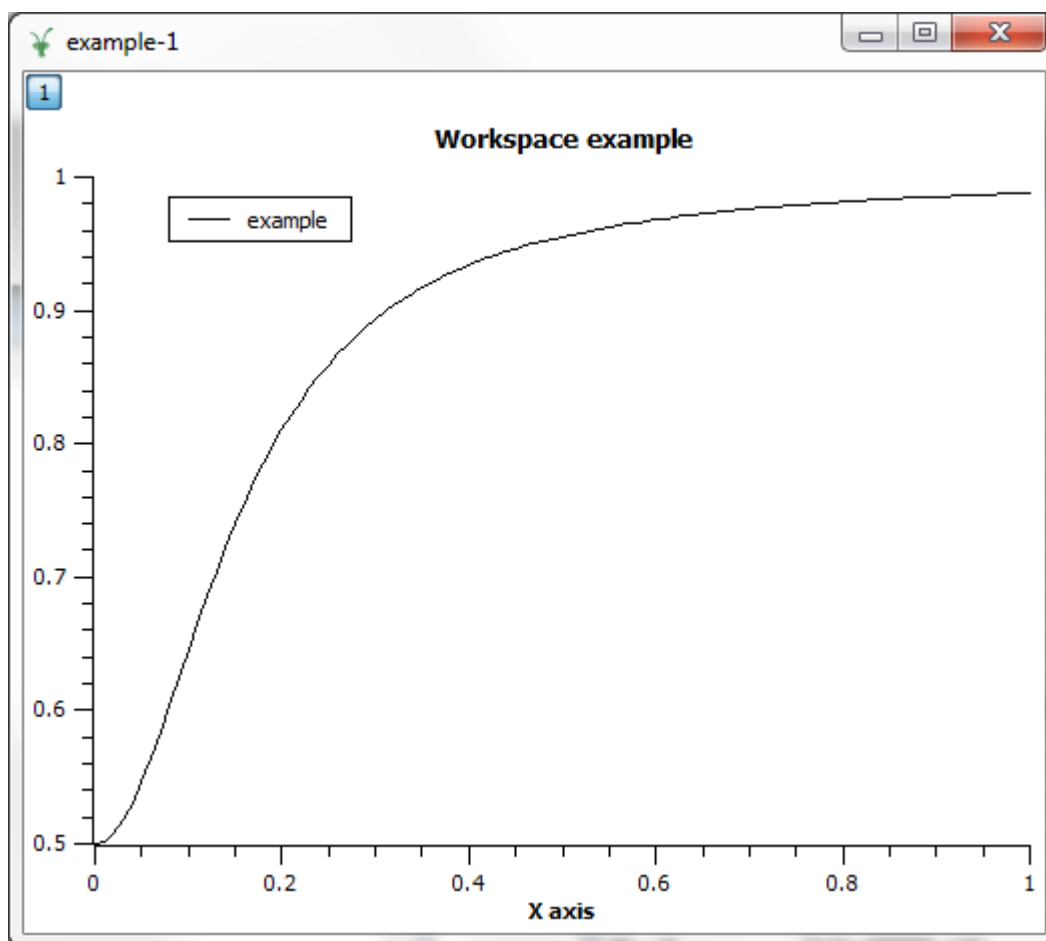
Line 1 says we have an isotropic hyperfine coupling of 4400 MHz between the muon and the electron.

Lines 2 and 3 mean sweep the magnetic field from 0 to 1 Tesla with 101 points (100G steps).

Line 4 means take the integral asymmetry (by default the integration time is 0 to infinity, weighted by the muon lifetime).



Executing the algorithm...



The result (this example gives a workspace with 1 spectrum).

Overview

Algorithm "QuantumTableDrivenSimulation" takes a control table which allows basically all the values that could be set in the original Quantum's dialog box.

There are no hard coded limits to the number of spins, sites, etc – only that the sizes of internal workspaces may increase exponentially and Mantid may run out of memory, or take an infinite time to complete the calculation.

A progress bar is displayed, and updated according to the loops being scanned. Cancelling the algorithm throws away the calculations done so far.

Quantum can act as a fit function, either a time spectrum or an integral curve versus field, etc. This can then be used in the Fit() algorithm or anywhere else in Mantid which does fitting, such as the Muon Analysis or Muon ALC interfaces. Fit function “QuantumTableDrivenFunction1” takes a control table as an Attribute, one parameter which is used to vary the model, and two additional parameters for scale factor and baseline. Functions “QuantumTableDrivenFunction2”, “QuantumTableDrivenFunction3”, etc. are similar but with more model parameters to vary.

Setting up

First, install Mantid if you don't already have it. Make sure you have a reasonably up-to-date version. <http://download.mantidproject.org/>

Open the Script Repository and download the files in the section Muon/Quantum, including the subdirectories.

In order to be able to run Quantum (or any user Mantid algorithm) from a script, at present it is necessary to load that algorithm at start up: do Preferences – Mantid – Directories and add the directory .../Muon/Quantum/Algorithms_Autoloaded to the box “Python extensions” and to “Python Scripts”. Then restart Mantid to make it load the algorithms. You probably want to do this anyway for convenience.

Alternatively you can just load the file “quantumtabledriven.py” into the Python Window and execute it. This will make the algorithms and fit functions available for use from the GUI.

Wizard mode

This is a simple way to get started. Just run the script “QuantumWizardScript.py”. It will ask questions in a series of dialog boxes – fill in the numbers and press “Run” for each. It will then run Quantum for you, generating the result as workspace “Results”, and plot it.

The wizard mode supports a limited set of the more common parameters. However it also generates the control table (named “Table”) which you can use as the basis for more advanced modelling, or edit if you want to re-run with only small changes.

Unfortunately you can't yet go back through the sequence of boxes if you realise you made a mistake earlier: if you cancel any of them, the script exits.

Control Table format

The usual input to Quantum is in a Mantid Table which should have two text columns, here referred to as “code” and “value” (the actual column names don't matter). A blank Mantid table can be generated by a Python script or algorithm (e.g. CreateFullTableWorkspace.py available from the Script Repository as above):

```
Tab=CreateEmptyTableWorkspace()
```

```

Tab.addColumn("str", "Code")
Tab.addColumn("str", "Value")

for i in range(30):
    Tab.addRow(["", ""])

```

With Mantid 3.0 to 3.3, table windows are not supposed to be editable but it does work – you have to double click the column headings and un-tick “Read Only”. However you cannot expand the table interactively, therefore the above script generates one that is “big enough”. It’s also hard to completely empty out a cell, the easiest way to remove an un-needed row is to comment it out with “#”.

Having filled in a table for a frequently used model, it can be saved and subsequently reloaded using Mantid’s standard save and load operations.

Rows where “code” starts with “#” are ignored (as are most unrecognised codes). Rows can be in any order.

Most of the codes can be omitted if not relevant or if the default value is suitable anyway. Keywords are case sensitive and generally in lower case (except element names and the Fit Function).

The examples below are written as “code=value”. (A text file format like this is also defined). Split the line into the two cells and do not include the “=” character. Note that some parameters such as “fitfunction” have additional “=” characters in the “value” part and these should be preserved.

Codes

Model definition

Code	Value/meaning	Example
spins	List the spins, which can be either an element (including “Mu” and e) which automatically defines I and gamma, or a number for I such as 1 or 5/2. Element names can have a number added to distinguish multiple atoms (H1, H2). Isotopes should have the atomic mass such as 7Li, 63Cu. When referring to these spins later, use the element name as given here or the number 0 to N in sequence.	spins=Mu,e,H spins=Mu,19/2 (interaction with large spin)
dynamic	Specify dynamic calculation mode. Value is the number of states/sites to convert between. Sites are referred later using “@0”, “@1”, etc or if not specified, the parameter applies to all sites.	dynamic=2
gamma(site,sp)	Gyromagnetic ratio in MHz/T, required if an anonymous spin I is given in the “spins” line. You can also specify a different value for a	gamma(H)=42.5

	named element.	
a(site,sp1,sp2)	Specify a hyperfine constant. "site" is optional. "sp2" defaults to the electron if not given. Forms for "value" are: A (isotropic) A,D (axial, symmetry axis defaults to along z) A,D,x,y,z (axial along specified axis) Axx,Ayy,Azz,Axy,Axz,Ayz (full tensor) Note – multiple A codes with the same spins perhaps in a different order just add up the hyperfine couplings.	a(Mu)=4463.0 a(@1,Mu,e2)=23.7,1.9,1,1,1
q(site,spin)	Quadrupole splitting. "site" is optional. Forms are: nuQ (axial, along Z) nuQ,eta (principal axis along x, other two along x and y) nuQ,x,y,z (axial, specified axis) Qxx,Qyy,Qzz,Qxy,Qxz,Qyz (full tensor, still in MHz. Should have Tr(Q)==0.)	q(63Cu)=1.7
r(site,spin)	Specify coordinates x,y,z in Å. Dipolar coupling will be calculated for all pairs of spins whose coordinates are both known.	r(F1)=1.15,0,0
relax(site,spin)	Relaxation rate in μs^{-1} for the spin given, in the specified site (or all)	relax(@0,e)=150.4
convert(site1,site2)	Conversion rate between specified sites, in μs^{-1} . Sites do not take the leading "@" character here, but ranges of sites can be given using "-"	convert(0,1)=53.8 convert(0-3,0-3)=2 (all possible transitions between 4 sites)
pop(site)	Initial population of site (scale 0 to 1). Populations will be normalised if all are given explicitly. Special value -1 for 2 or more sites means these sites are set to the equilibrium populations based on the remaining total population and the conversion rates among them (but not to/from any fixed sites). -2 for one or more means share out the remaining population equally among these sites. (-1 and -2 simultaneously is not meaningful, the -1 sites are filled first)	pop(@0)=0.3
pulsed	Activate pulsed (time sliced) mode. Specify the times at which the Hamiltonian will change. Elsewhere use the syntax a(1,Mu) for A,q,B, etc. Omitting n means that value applies at all times. For T time slices (T-1 times given) and D sites (D=1 for non-relaxing), define T*D values using (t,@d,spin).	pulsed=2.0,2.1 (0.1 microsecond pulse)
tzero	Activates finite pulse rounding, specifies the pulse shape(s) to convolute with (g=Gaussian, l=Lorentzian, p=parabolic,	tzero=p,0.070,e,0.026 (parabolic proton pulse and pion lifetime)

	u=uniform, e=exponential, c=half cosine) and width. 1 st time slice (if sliced) initial density matrix is effectively tracked forward or back by random amounts using its Hamiltonian before any points are sampled. Done by integration of density matrix components. Note RF phase of this slice is therefore locked to random arrival time and not t=0 as for others. (Could average this too?)	tzero=u,0.01,u,0.01 (triangular pulse with rise and fall time = 0.01 us)
--	--	--

Measurement and averaging

Code	Value/meaning	Example
lfuniform	Number of orientations, which are arranged uniformly distributed over the sphere. Field, initial spin and detector are all parallel.	lfuniform=100
tfuniform	As lfuniform but in Transverse geometry with the field perpendicular to the initial spin and detector parallel to initial spin	tfuniform=50
lfrandom	Number of Monte Carlo random orientations	lfrandom=1000
tfrandom	Number of Monte Carlo random orientations	tfrandom=200
lf	Specify single axis for field and initial spin (no averaging). Default lf=0,0,1 if no averaging options are given.	lf=1,1,0
tf	Specify axes for Transverse geometry (no averaging): Bx,By,Bz,Sx,Sy,Sz (2 axes, field then beam/detector/RF) Bx,By,Bz,Sx,Sy,Sz,Dx,Dy,Dz (3 axes, field, beam/RF, detector) Bx,By,Bz,Sx,Sy,Sz,Dx,Dy,Dz,Rx,Ry,Rz (all 4 specified, useful for longitudinal RF too)	tf=1,0,0, 0,1,0
lfaxes	Specify a single axis which will be permuted into all possible equivalent directions in cubic symmetry, removing duplicates and opposing axes.	lfaxes=1,0,0 (which will measure (1,0,0), (0,1,0) and (0,0,1) and can give an exact zero field powder average)
bmag(site)	Specify the magnitude of the magnetic field (Tesla). With 3 values, vector field along (nominal field axis, nominal RF B1 axis, 3 rd mutual perp axis).	bmag=0.002 bmag(1)=0.001,0.05,0.0 (pulsed TF in addition to small LF)
bmagGauss(site)	Specify the field in Gauss.	bmagGauss=1500.0
measure	Measurement type: "timespectra", "integral", "fit", "m0", "m1", "m2", "freqspec", "breitabi"	measure=fit
starttime	Starting time for the measurement or generated spectrum (defaults to 0.0)	starttime=1
endtime	Ending time for the measurement. Defaults to 20μs for time spectra and fitting, "Inf" for integral counting.	endtime=2.0
minfreq	Lower limit (MHz) for a frequency spectrum, or for moment calculations. Give 2 numbers to	minfreq=0

	calculate the value based on bmag: $f=A+B \cdot \text{bmag}$	
maxfreq	Upper frequency limit (MHz)	maxfreq=30
ntbins	Number of time bins to generate in a time or frequency spectrum (Default 1000 for time spectra and fitting, 1 for integral counting).	ntbins=200
fitfunction	The analytical function to fit to the generated spectrum. Passed to Mantid's "Fit" algorithm as the "Function" property, can use User Function, initial values, etc.	fitfunction=name=GausOsc fitfunction=name=UserFunction, Formula=a+b*x,a=1,b=2
mulife	Muon lifetime, for integral counting (default 2.19703 μ s) or fit data point weighting (default all points equally weighted)	mulife=2.19703
fitstats	Number of Mevents to weight error bars when fitting. With mulife set, refers to whole time range 0 to inf. Without mulife, refers to requested time window only. Assumes error proportional to $1/\sqrt{\text{average count rate}}$ with no random scatter, and asymmetry=0.25.	fitstats=20.0
recycle	Reuse the listed fitted values (only) as the initial values for the next loop point. These initial values must have been provided in the "fitfunction" string. If the fit function contains more than one component, the names must be prefixed with "f0", "f1", etc just as in a Parameters table or the spectra names in the results.	recycle=f0.Frequency
initialspin	Which spin to set to fully polarised (default the first one)	initialspin=e
detectspin	Which spin's polarisation is detected (default the first one)	detectspin=H
brf(site)	RF B-field, in Tesla, along the direction as specified above. Specify magnitude (peak if linear), frequency, phase (default 0), ellipticity (default 0=linear) and RRF index (default 0) With no brf code given, RF mode is not used. In time slice mode brf need not be given for all slices (site). Phase is in degrees, or use "r" for random phase, different for every orientation. Ellipticity=1 means circularly polarised RF field (constant magnitude as given). RRF=1 means reference frame rotating with RF (effectively rotating detector).	brf=0.0001, 25.0, 90.0

Loops

Code	Value/Meaning	Example
loop0par	Code to vary on the "inner" loop. Can be just the code itself, if it takes a single value (and in this case the code need not be separately listed with a fixed value). Code[i] varies one number in a list of	loop0par=bmag (simple field scan) loop0par=a(Mu)[1] (varies D)

	values (starting from 0). Code[i,j] scans values i and j round the unit circle, where the loop variable is in degrees. Multiple values can be scanned simultaneously, separated with “;”	loop0par=lf[0,1] (rotates field in x-y plane) loop0par=r(F1)[2];r(F2)[2] (moves two atoms along z)
loop1par	As loop0par for the outer loop, if used	loop1par=relax(e)
loop0range	Start value, end value, number of points. If more than one parameter, repeat the start and end for each (steps is always last). A negative number of steps means scan on a log scale (entirely negative ranges are allowed here)	loop0range=0,0.5,101 loop0range=1,1.5,-1,-1.5,51
loop1range	As loop0range for the second loop.	loop1range=0.1,1000,-5 (values will be 0.1,1,10,100,1000)

As a Fit Function:

Code	Value/Meaning	Example
measure	Only values “timespectra” or “integral” make sense.	measure=timespectra
ntbins	For measure=timespectra, values of ntbins, starttime and endtime are ignored and the X values of the data used instead.	
loop0par	For measure=integral, (or 2D fits?), gives the code to vary according to the data’s X axis	loop0par=bmagGauss (to fit a field scan directly from PlotAsymmetryByLogValue)
loop0range	For measure=integral, the values here are ignored and the data’s X axis is used instead.	
fit0par	Code to use as the first fit parameter (named P0 in the Fit Function Browser). As for loops, it can be just the code itself, if it takes a single value (and in this case the code need not be separately listed with a fixed value). Code[i] varies one number in a list of values (starting from 0). Code[i,j] scans values i and j round the unit circle, where the loop variable is in degrees. Takes multiple values separated by “;” and a code can be prefixed with “-” to be set to the negative of the fit parameter.	fit0par=a(Mu) (simple fit of repol curve) fit0par=a(Mu)[1] (varies D) fit0par=r(F1)[2];-r(F2)[2] (symmetric molecule stretch)
fit1par	As fit0par for the second parameter (P1). fit2par, fit3par, etc also available.	fit1par=relax(e)

Outputs

The output is generally a Workspace2D.

If “Time Spectra” mode is selected, the X axis is the time bins from the simulated spectra. The Y axis may be the loop if used, and it will have a numeric axis suitable for Colour Fill Plot, otherwise the workspace only has one spectrum.

If “Fit” measurement mode is selected, the individual spectra correspond to the fit parameters (plus chi-squared), complete with errors. The Spectra axis has the names of these parameters so “Plot spectra” labels them sensibly. The loop (if used) is along what would be the time (X) axis of the workspace. If the “recycle” parameter is given, some fitted values are reused as starting points for subsequent points in the same workspace, thus tracking a varying frequency for example.

In “Integral” mode two loops can be used and “Colour Fill Plot” may be used. With one loop, the workspace only has one spectrum.

The moment calculations “m0”, “m1” and “m2” look for frequencies in the specified range (which can shift with the applied field). m0 is the total amplitude in the range. m1 is the mean frequency, weighted by amplitude. m2 is the second moment or linewidth. Two loops may be used.

“freqspec” generates a frequency spectrum (along X) over the specified range, using the absolute frequencies and amplitudes, and ignoring phases or relaxation if present. If the range is set to vary with field, the X axis corresponds to field=0, i.e. offsets from the Larmor frequency.

“BreitRabi” mode outputs one spectrum per eigenstate. The loop (X axis) usually corresponds to magnetic field. Levels are sorted in order of energy and therefore assumed to avoid each other if the scan is too coarse to show the detail at a crossing. No orientation averaging or dynamics should be used.

Loops scanned on a log scale are probably best plotted on a log scale too (plot options dialog). This also works for axes of 2D plots.

When using as a fit function, do not use any of the “random Monte Carlo” options such as lfrandom, as the resulting small variations from run to run can upset the fit minimiser.

After fitting, the optimum values for the parameters are NOT automatically copied back into the model table. You may want to do this manually, or there is a helper algorithm “ReplaceFitParsInTable” for this. Pass it the original table and the _Parameters result from the fit, and if you’ve used a composite function you may need to hint which component was the Quantum function (others may have similar parameter names).

The table used for a fit function, if passed to “QuantumTableDrivenSimulation” instead, should generate a simulated data set which can be overlaid on the data, except with user specified time bins or loop range.

Time sliced mode

At each time slice point the final density matrix is calculated and used as the starting value for the next block. If in dynamic mode with multiple sites they are mapped 1:1 on change over.

For pulsed RF:

```
Pulsed=1.0,2.84          # 1.84us pulse (90 deg for diamagnetic muons
if B1(lin)=20G, B1(rrf)=10G)
brf(|0)=0,0             # RF off to start with (or don't specify it). Either
way, uses plain calc for this interval.
```

```
brf(|1)=0.0020,25.0,0.0      # 20 Gauss (peak) at 25 MHz, 0 deg phase
                              (projected back to t=0)
brf(|2)=0,0                  # RF off again. If still on, need not have same
                              frequency!
```

For laser excitation:

```
Pulsed=1.0,1.007 # 7ns pulse
Dynamic=2
a(@0,Mu)=500
a(@1,Mu)=100 # these hyperfine consts do not vary with time
pop(@0)=1.0   # all in ground state to start
convert(1,0)=1.0 # relaxation, excited state lifetime 1 us, any time
convert(|1,0,1)=1000.0 # excitation by laser (mean excitation
                        time 1ns here, vary according to intensity)
```

In general, can include (|slice) along with @site for a, gamma, r, q, relax, etc and also |slice alone for bmag, brf, convert.

Notes

Fields are normally in Tesla. If fitting an ALC curve you can use loop0par=bmagGauss to work in Gauss, especially from within the ALC Interface. Alternatively convert the X axis of your experimental spectrum with ScaleX.

Times are in microseconds which will match the X axis of raw data sets.

Scripting

It is possible to run QuantumTableDrivenSimulation from a Python script, just like any other Mantid algorithm. If you just look at the History of the output workspace, it will have something like:

```
CreateEmptyTableWorkspace(OutputWorkspace='Tab')

QuantumTableDrivenSimulation(ModelTable='Tab', Results='zfmuonium')
```

You will need to insert the missing code from “Table editing” above, with lines such as

```
Tab.addRow(["spins", "Mu,e,H"])
```

to fill in the table. You can of course use variables, but remember the table is in text format:

```
Tab.addRow(["convert(0,1)", str(MyConvRate)])
```

Alternatively you could Load the table from a file and just make any necessary changes.

Finally you can run the algorithm with

```
Results = QuantumTableDrivenSimulation(ModelTable=Tab)
```

where Tab and Results are now Python variables pointing to the table and the result workspace, not the names: these workspaces might not even appear in the workspace list (Analysis Data Store) if this code is within a parent Algorithm.

See the Mantid documentation for how to automatically plot the spectra from the workspace, etc.

Lower level code

If you're not familiar with Python, stop reading now!

It is possible to write a Python script to call the various internals of Quantum directly, either to run a complex simulation faster or to do something not yet possible via the table.

Start with:

```
from quantumtools import *
```

You'll probably also want to import `numpy`, `math`, and various Mantid packages.

Then set up the basis states for your problem:

```
spmat=createSpinMat([2,2])
```

where the argument is a list of values of $(2I+1)$ for the spins to be modelled. Here there are two spin-1/2 spins. The array `spmat` or slices taken from it are used below.

Define the initial polarisation direction, and the detector operator:

```
rho0 = createInitialDensMat(spmat[0],beam)
```

where `beam` is a 3 element list or array in the direction of the initial polarisation (need not be normalised). The array index `[0]` for `spmat` means that spin 0 will be set to fully polarised and the others unpolarised.

```
scint = createDetectorOp(spmat[0],bank)
```

where `bank` is another 3-element array giving the direction to the detector. Here spin 0 is detected.

Define a blank Hamiltonian to which various interactions will be added:

```
Ham=numpy.zeros_like(spmat[0,0,:,:])
```

Zeeman interaction (you will need one per spin):

```
addZeeman(Ham, spmat[0], B, 135.5*Bmag)
```

here we add a Zeeman interaction for spin 0, `B` is a vector (which will be normalised) giving the direction of the field and $(135.5*Bmag)$ is the Larmor frequency in MHz (which would be negative if γ is negative or the true `B` is opposite to the direction given).

```
addHyperfine(Ham, spmat[0], spmat[1], AxialHFT(A,D,[0,0,1]))
```

Add a hyperfine (or exchange...) interaction between spins 0 and 1. The 4th parameter is the hyperfine tensor: instead of giving 9 components (a 3*3 numpy array) you can use the short cut function `AxialHFT(A,D,axis)` or just give a single value for an isotropic interaction.

```
addDipolar(Ham, spmat[0], spmat[1], r1, gamma1, r2, gamma2)
```

Dipolar interaction. `r1` and `r2` are 3-vectors giving the location of the two spins in Å. `gamma1` and `gamma2` are the gyromagnetic ratios (MHz/T).

```
addQuadrupole(Ham, spmat[1], nutensor)
```

Quadrupole interaction for a spin with $I > 1/2$. The full tensor can be given or you can use `AxialHFT(0,nuQ,[1,1,0])` for example.

This is now sufficient to solve a simple model without any dynamics or RF resonance. Use:

```
(omega, ccos, csin) = solveDensityMat(Ham, rho0, scint)
```

This returns the solution as a series of coefficients. Note that there may be degenerate values especially in zero field. The solution $y(t) = \sum(ccos[i]*\cos(\omega[i]*t) + csin[i]*\sin(\omega[i]*t))$ is then evaluated using:

```
y = evaluateIntoBins(omega, ccos, csin, lam, times)
```

where `lam` is the inverse muon lifetime for averaging purposes and `times` is an array of $n+1$ time bin boundaries in microseconds, perhaps taken from the `X` values of a Mantid workspace. `y` is then an array giving the (weighted) average polarisation in each of the n intervals. For integral counting pass `times=[0,float("inf")]`. There are two versions of this function: `evaluateIntoBins` is optimised for lots of time bins and a fairly simple model while `evaluateIntoBinsNewLoops` is better for integral counting of complex systems.

To average over orientations, you need an outer loop. There are some iterators which may be helpful. They follow the form:

```
for (field, beam, detector, rf) in uniformLF(N):
```

These return four 3-vectors for the directions of the magnetic field, initial polarisation, detector, and RF magnetic field. `uniformLF` and `uniformTF` give N orientations equally spaced in $\cos(\theta)$. `randomLF` and `randomTF` generate N random orientations (different every time). `uniformRF` and `randomRF` are for RF 90 degree precession with the muons parallel to B but the detector perpendicular. If you need a 3rd mutually perpendicular axis, calculate it with `detect90=numpy.cross(field,detector)`.

For modelling of dynamic systems, proceed as above to define a Hamiltonian for each state (if more than one). Then:

```
BigHam, SmallHamSize, nsites=buildDynamicHam([Ham1, Ham2])
```

Assembles the list of individual Hamiltonians into a composite one. Use [Ham] if there is only one, to which spin relaxation might be added. nsites is the number of states given.

```
addSpinRelaxation(nsites,BigHam,state,spmat[1],nu,pol,polmag)
```

add an “artificial” relaxation of spin 1 when in site “state”, at rate nu. pol is either a mini density matrix for this spin once in its relaxed state, or a 3-vector indicating the direction of the residual polarisation. Polmag is a scale factor (0 = unpolarised, i.e. complete depolarisation, 1.0=polarised as specified by pol). Pass 0.0 for both for simple depolarisation.

```
addConversion(nsites,BigHam,state1,state2,nu,rsps,pols)
```

nsites is the number of states, the conversion is from state1 to state2 (integers) at rate nu (inverse microseconds). rsps and pols will define spins to be depolarised on conversion, and their final polarisations, in a similar manner to addSpinRelaxation (not yet implemented so pass 0.0).

```
BigRho=createDynamicDensMat(rho0,pops)
```

Create the composite initial state. rho0 is the value from createInitialDensMat above, and pops is an array of length nsites giving the population of each state (should be normalised). You need this even for a 1-site problem, pass pops=[1.0].

```
BigScint=createDynamicSpinOp(scint,nsites)
```

Create the composite detector operator. scint is the 1-site operator as above. The detector cannot distinguish between sites (but you might be able to build your own operator which does)

```
(omega,ccossin)=solveDynamicResult(BigHam,BigRho,BigScint)
```

Solve the dynamic system. The solution will be $y(t) = \text{Re}(\sum(\text{ccossin}[i] * \exp(\omega[i]*t)))$. omega and ccossin are complex arrays, and $\text{Re}(\omega(i))$ should always be zero or negative indicating relaxation rather than growth.

```
y = EvaluateDynamicIntoBins(omega,ccossin,lam,times)
```

Evaluate the dynamic solution, similar to evaluateIntoBins().

RF resonance:

Define a static Hamiltonian. Also define a RF Hamiltonian which is usually the Zeeman splitting in the peak value of B1. For rotating fields there will be two RF Hamiltonians at 90 degrees phase. The total instantaneous Hamiltonian is $\text{StaticHam} + \text{RFHam0} * \cos(\omega_{\text{RF}} * t) + \text{RFHam90} * \sin(\omega_{\text{RF}} * t)$.

For rotating reference frame detection at the applied frequency, define a second detector operator with `scint90 = createDetectorOp(spmat[0],bank90)`.

Then:

```
(omega,ccos,csin) =
solveRFDensityMat(StaticHam,RFHam0,RFHam90,omegaRF,rho0,scint0,scint
90,RRFharmonic)
```

For simpler cases pass RFHam90=None, RRFharmonic=0, scint90=None.

The values of omega will be limited to the range $(-\omega_{RF}/2, \omega_{RF}/2)$ and the answer is only strictly correct at times which are a multiple of the RF period. However a smooth curve can still be interpolated. Use evaluateIntoBins() as before.

Time sliced mode:

Each of solveDensityMat(), solveDynamicResult() and solveRFDensityMat() takes an optional extra named parameter timeend=t. They then return a 4th result which is the density matrix evaluated at this time. This can be used instead of rho0 (or BigRho) in the next time slice.

Higher level functions may be useful:

```
y = EvaluateSliced([Ham0,Ham1,Ham2],rho0,scint,[t01,t12],times,lam)
```

Processes a time sliced sequence without RF and evaluates the result, taking care where time bins do not coincide with slice boundaries.

```
y =
EvaluateSlicedDynamic([BigHam0,BigHam1,BigHam2],BigRho0,BigScint,[t0
1,t12],times,lam)
```

Processes a time sliced sequence with dynamics.

```
y =
EvaluateSlicedRF(Hams0,HamsRF,HamsRF90,omegasRF,rho0,scint,scint90,
RRFharmonic,slices,times,lam)
```

The RF version, for RF pulse sequences. Hams0=[Ham0,Ham1,Ham2,...].

HamsRF=[RFHam0,RFHam1,RFHam2,...]. HamsRF90=[RFHam90,RFHam90,RFHam92,...].

omegasRF=[omegaRF0,omegaRF1,omegaRF2,...]. slices =[t01,t12,...]