Quantum for Mantid “table driven” instructions

What is Quantum?

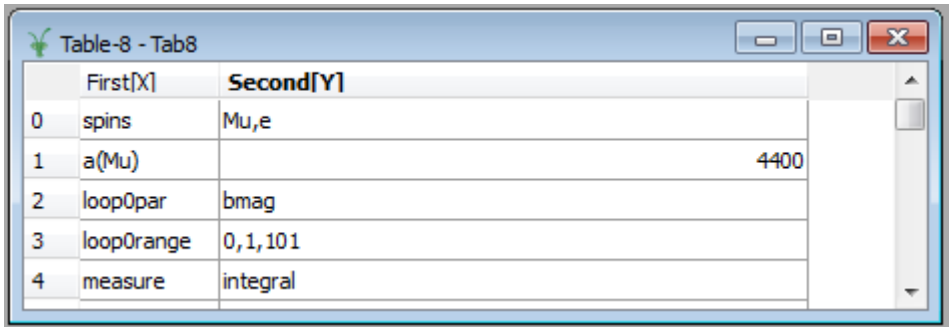
A program to solve the time evolution of the muon spin, given various interactions such as dipolar, hyperfine or quadrupole splitting, and applied magnetic fields. Using fitting, it can work back from experimental data to find the unknown interactions. It uses the density matrix method.

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:

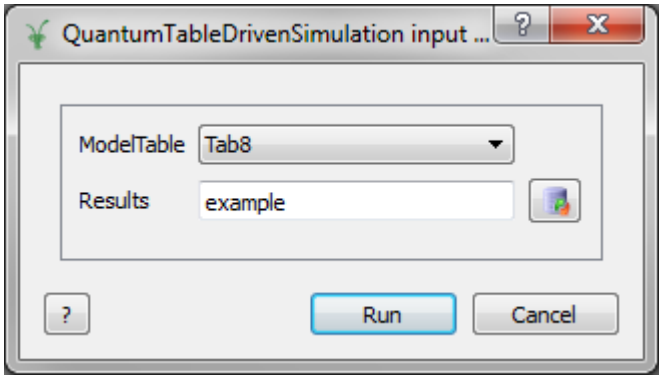


Line 0 defines the spins involved, and implicitly sets the gyromagnetic ratios.

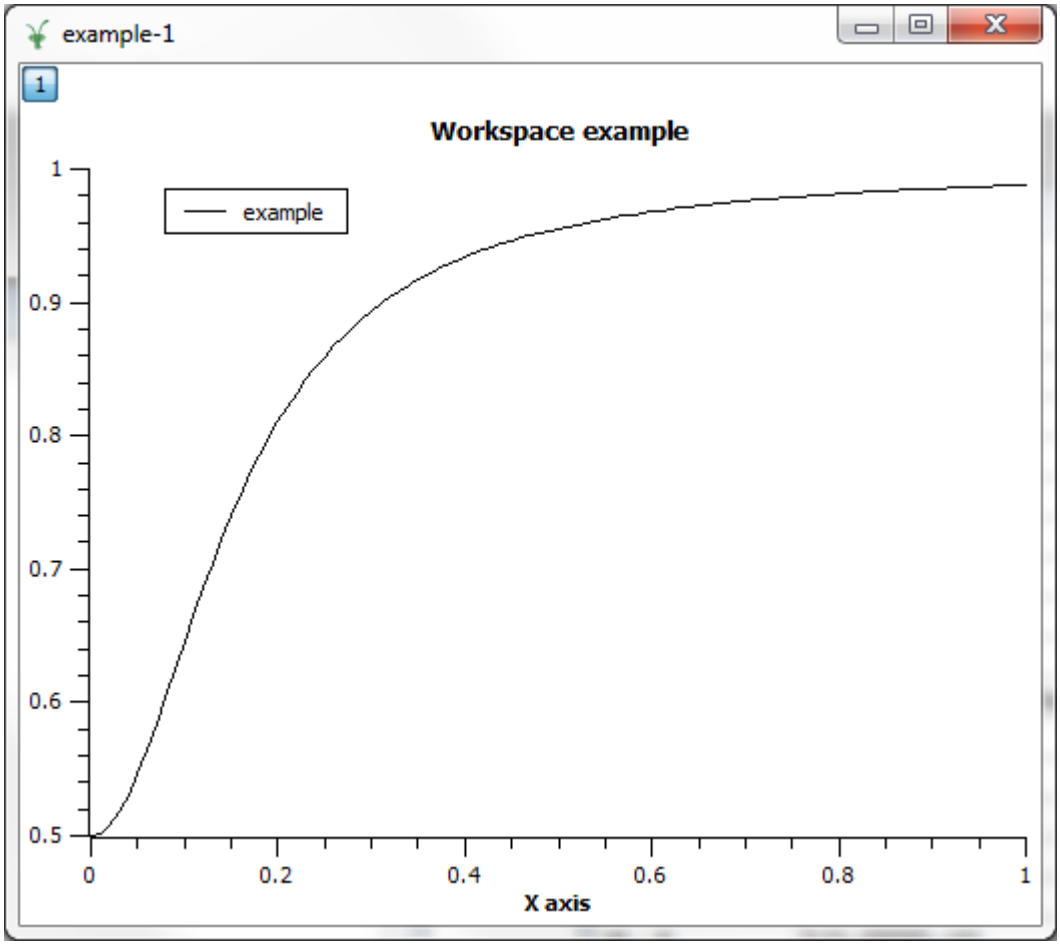
Line 1 says we have an isotropic hyperfine coupling of 4400 MHz between the muon and the electron.

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).



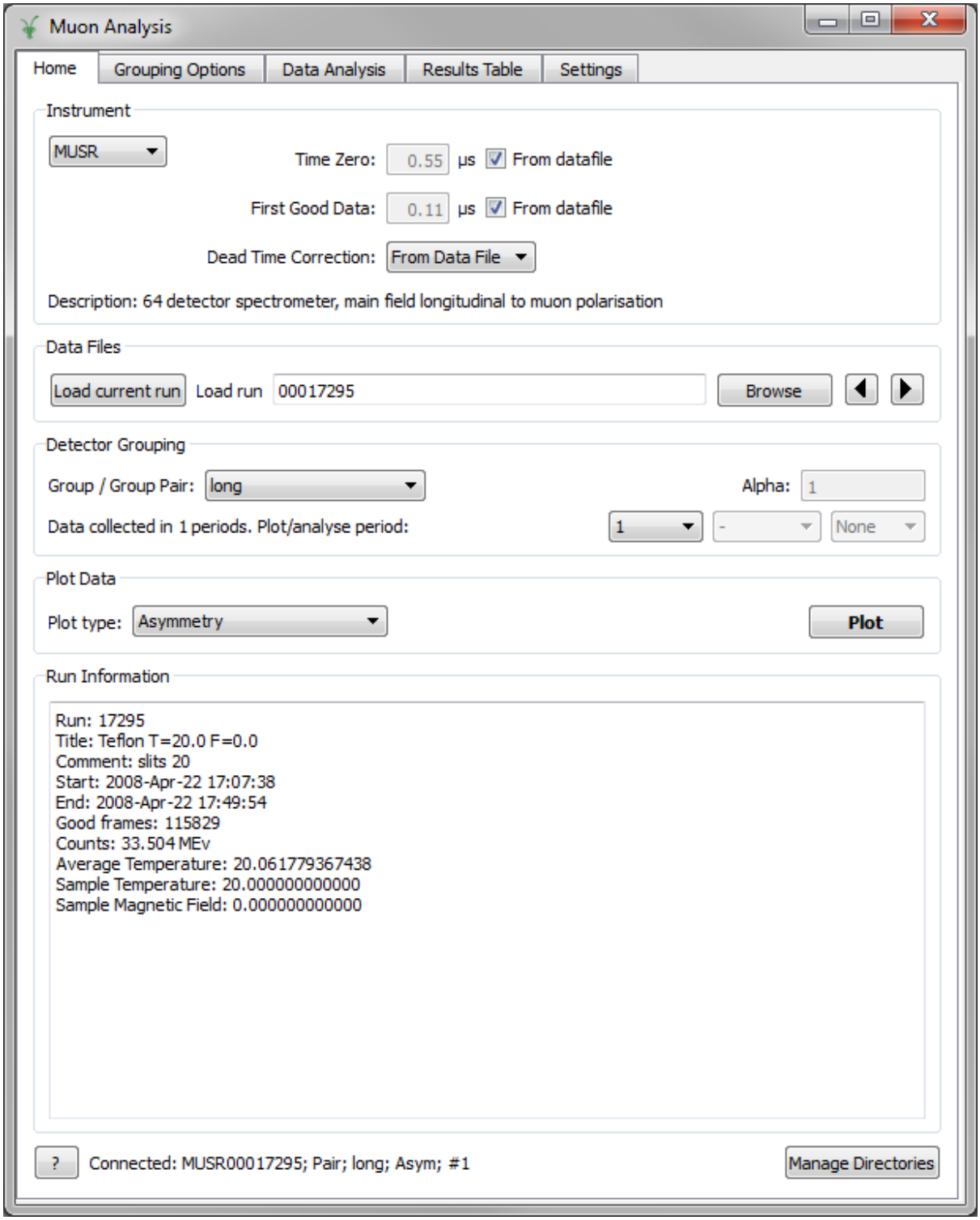
Executing the algorithm…

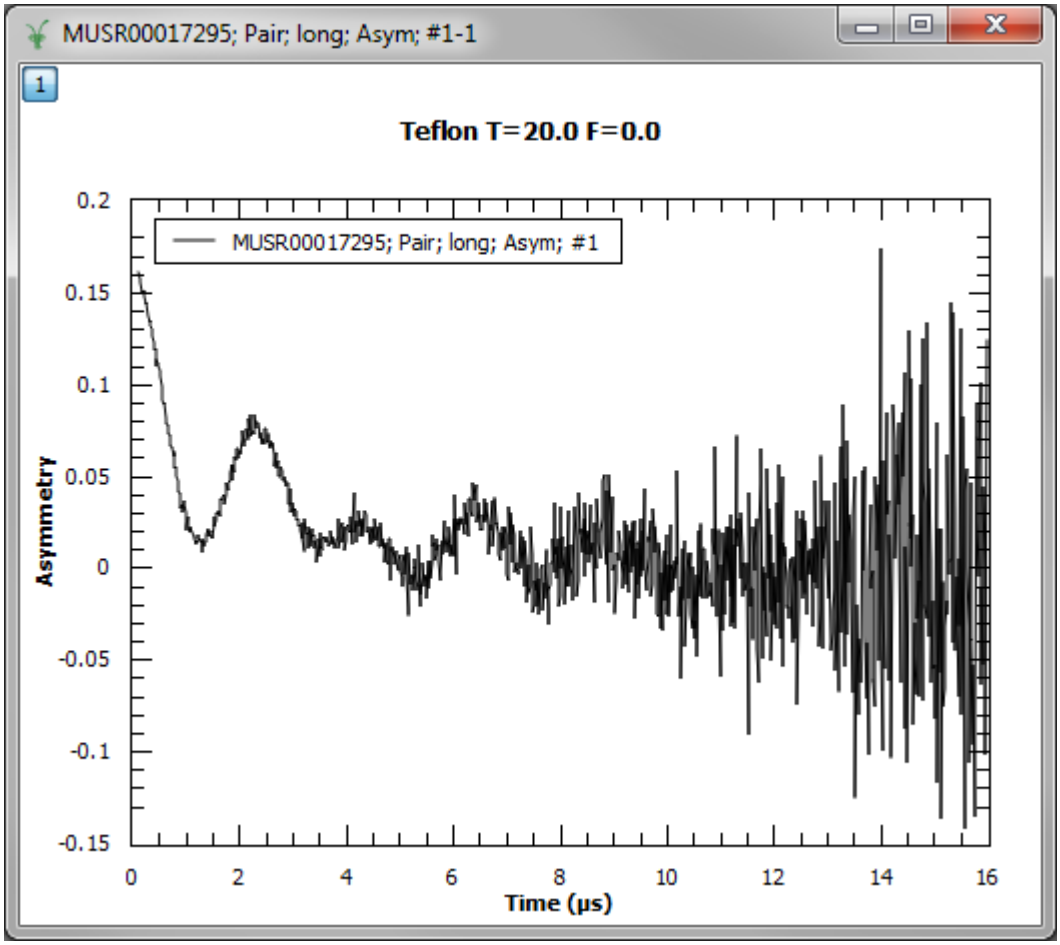


The result (this example gives a workspace with 1 spectrum).

Fitting example: F-mu-F

Load the run in the Muon Interface


Muon Analysis
Home
Grouping Options
Data Analysis
Results Table
Settings
Instrument
MUSR
Time Zero:
0.55
μs
From datafile
First Good Data:
0.11
μs
From datafile
Dead Time Correction:
From Data File
Description: 64 detector spectrometer, main field longitudinal to muon polarisation
Data Files
Load current run
Load run
00017295
Browse
Detector Grouping
Group / Group Pair:
long
Alpha:
1
Data collected in 1 periods. Plot/analyse period:
1
-
None
Plot Data
Plot type:
Asymmetry
Plot
Run Information
Run: 17295
Title: Teflon T=20.0 F=0.0
Comment: slits 20
Start: 2008-Apr-22 17:07:38
End: 2008-Apr-22 17:49:54
Good frames: 115829
Counts: 33.504 MEv
Average Temperature: 20.061779367438
Sample Temperature: 20.000000000000
Sample Magnetic Field: 0.000000000000
?
Connected: MUSR00017295; Pair; long; Asym; #1
Manage Directories



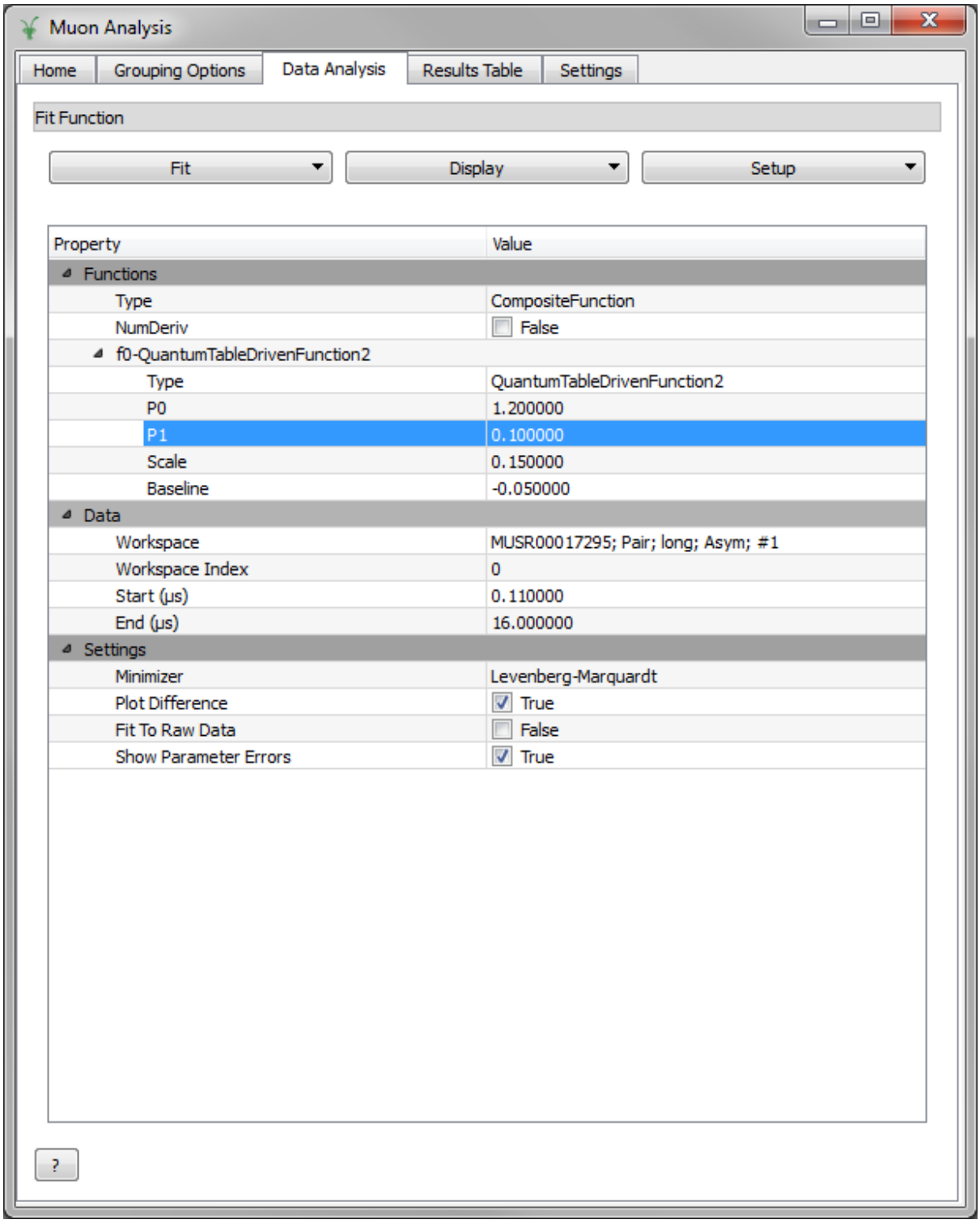
Define the model. You must call the table “Tab”.

Table-1 - Tab

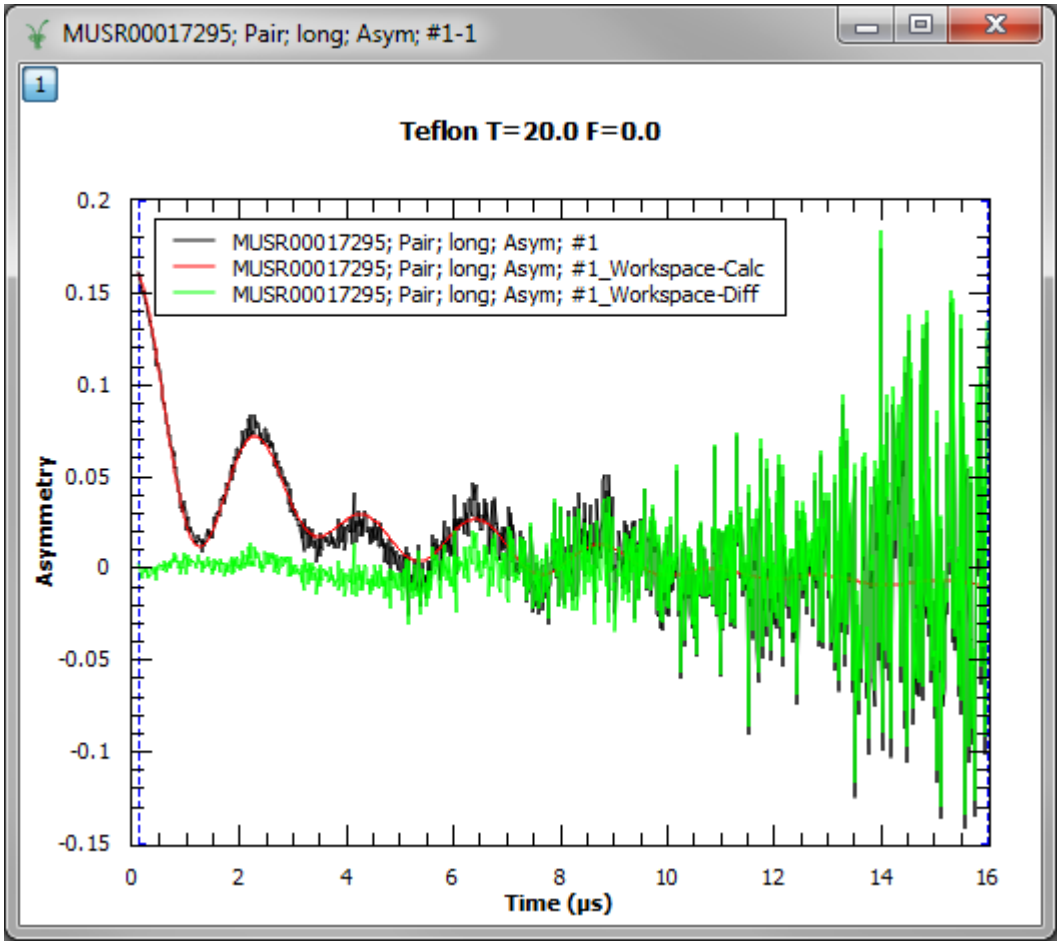
	Code[Y]	Value[Y]
0	spins	Mu,F1,F2
1	r(Mu)	0,0,0
2	r(F1)	0,0,1.2
3	r(F2)	0,0,-1.2
4	lfuniform	50
5	bmag	0
6	measure	timespectra
7	fit0par	r(F1)[2];-r(F2)[2]
8	fit1par	relax(F1);relax(F2)
9	dynamic	1
10	relax(F1)	0.1
11	relax(F2)	0.1
12		
13		
14		

(You can run QuantumTableDrivenSimulation on this directly to check if it looks about right)

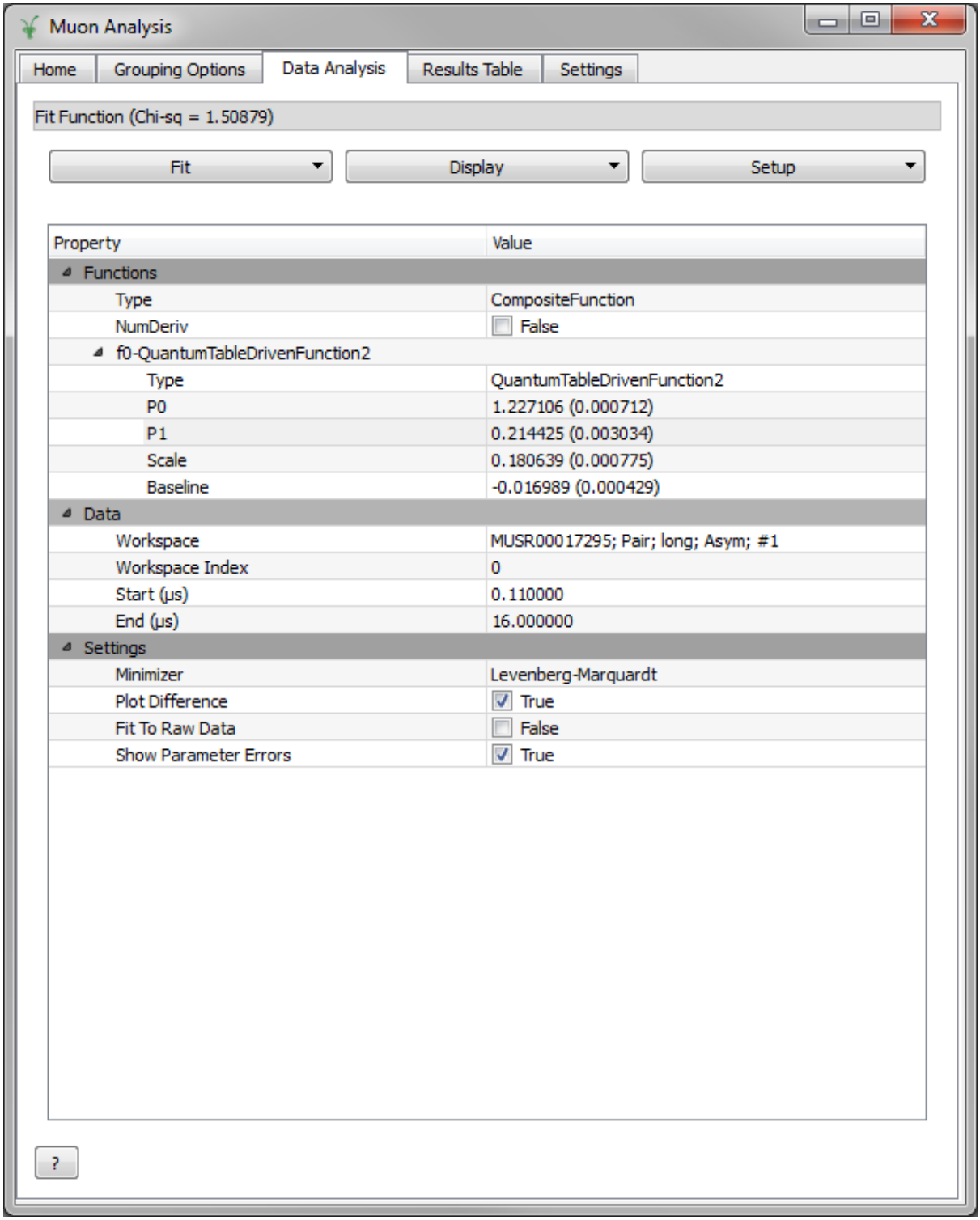
Select the Fit Function in the Muon interface. This model has 2 variables as defined by fit0par and fit1par so use the appropriate function. Give approximate starting values.



Fit!

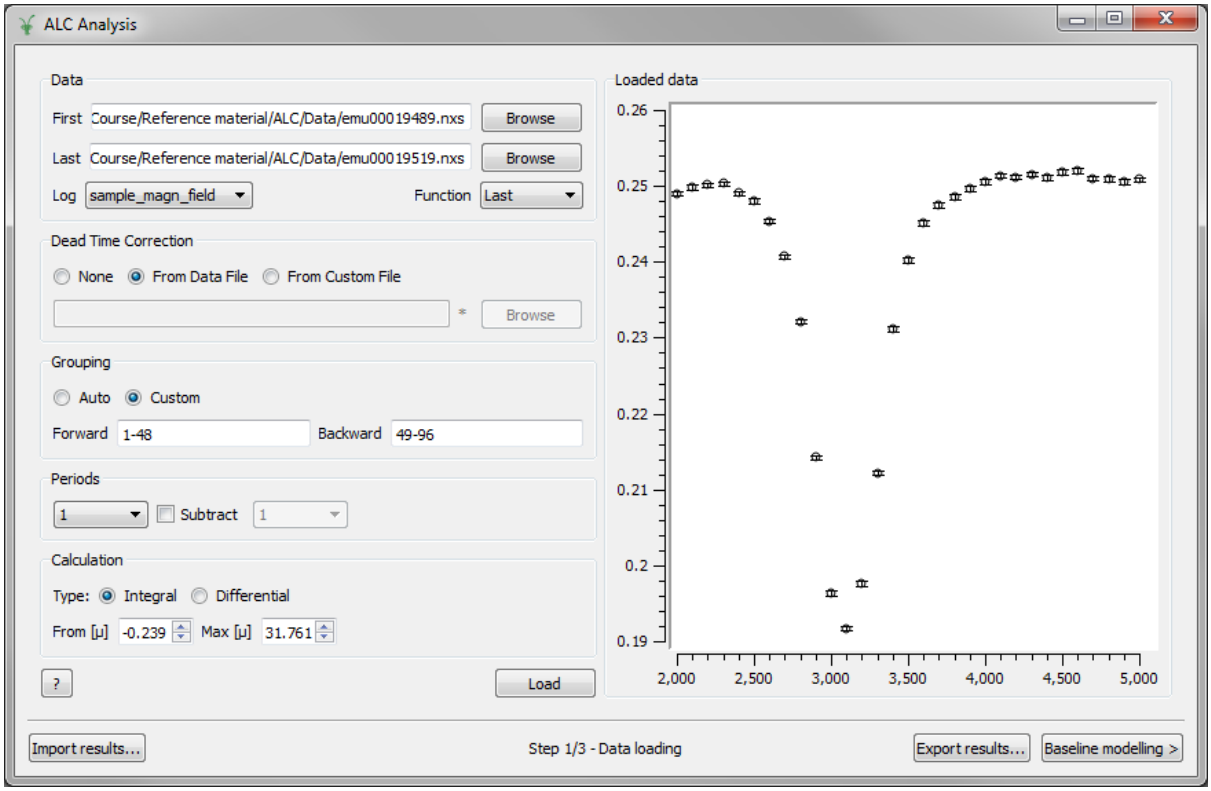


Completed fit with error bars:



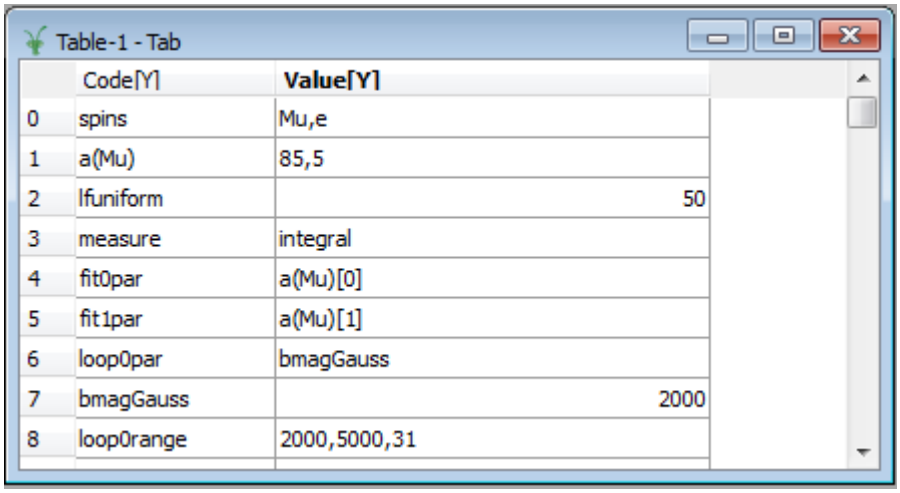
Fitting example 2 – ALC

Load some data:



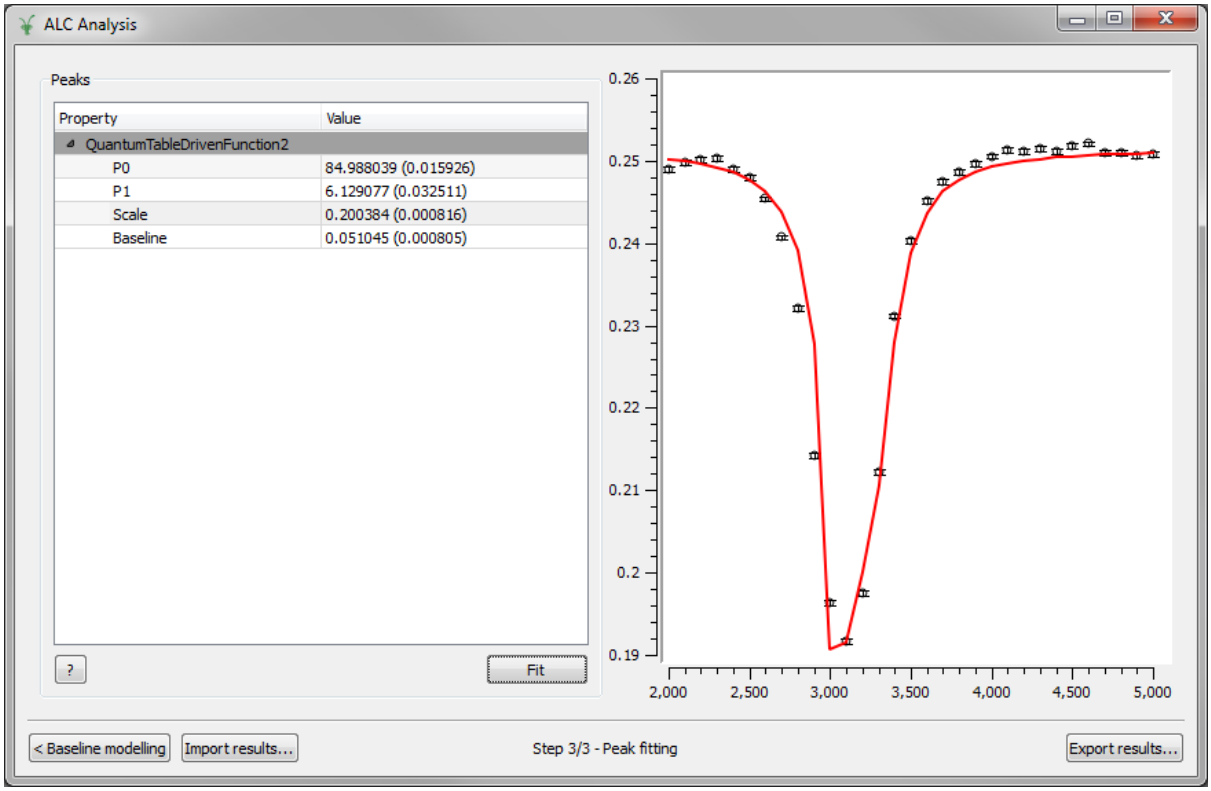
You may need to use the Background modelling, but it is not necessary for the baseline to be at 0.0.

Set up the Table:



	Code[Y]	Value[Y]
0	spins	Mu,e
1	a(Mu)	85,5
2	lfuniform	50
3	measure	integral
4	fit0par	a(Mu)[0]
5	fit1par	a(Mu)[1]
6	loop0par	bmagGauss
7	bmagGauss	2000
8	loop0range	2000,5000,31

Use the function:



These examples and others are available to download along with Mantid, see below.

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, 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, in that session only.

Examples and Tests

The directory "…/Muon/Quantum/Examples" contains a series of example control tables for Quantum including the ones shown above. There is also some test data to fit. The script "UnitTester.py" in that directory will run through all the examples and check that the results match the ones calculated earlier, to confirm that Quantum is working correctly.

Wizard mode

This is a simple way to get started. Just load the script "QuantumWizardScript.py" into the Python Scripting window and Execute it. 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. CreateBlankTableForQuantum.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. Rare earth f-electron spins can be entered by “f” plus the element name, e.g. fHo. These have the correct J and g-factor. 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 named element. Anisotropic gamma values use principal values as for g and a.	gamma(H)=42.5
g(site,sp)	Gyromagnetic ratio expressed as g-factor	g(e)=1.7,1,1,1,1.8,1,-1,0,1.9

	(multiple of Bohr magneton)	
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,E (non-axial, axes default to z,x,y) A,D,x,y,z (axial along specified axis) Axx,Ayy,Azz,Axy,Axz,Ayz (full tensor) A1,x1,y1,z1,A2,x2,y2,z2,A3 (principal values and axes: x3,y3,z3 are assumed perpendicular to other two) 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 z, 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.) Q1,x1,y1,z1,Q2,x2,y2,z2,Q3 (principal values and directions, as for A)	q(63Cu)=1.7
bNN(spin)	Crystal field coefficients b20,b21,…b66. Also ib21..ib66 for imaginary parts.	b22(fHo)=0.3
crystal	Specify space group symbol and unit cell a,b,c,alpha,beta,gamma. Used for coordinates, and crystal symmetry iteration	crystal=Fd-3m,4.2,4.2,4.2
r(site,spin)	Specify coordinates x,y,z in Å (or as fraction of unit cell if "crystal" given). 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	pop(@0)=0.3

	sites are filled first)	
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, u=uniform, e=exponential, e0=exponential with time offset to get phase=0, et=exponential truncated at 2 lifetimes, c=half cosine) and width. 1st 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=p,0.052,e0,0.026 (model ISIS: parabolic proton pulse and pion lifetime) 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
pquniform	As tfuniform, but the detector is rotated by 90 degrees round the field axis from the initial spin direction	pquniform=100
lfrandom	Number of Monte Carlo random orientations	lfrandom=1000
tfrandom	Number of Monte Carlo random orientations	tfrandom=200
pqrandom	Number of Monte Carlo random orientations	pqrandom=500
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	lfaxes=1,0,0

	into all possible equivalent directions in cubic symmetry, removing duplicates and opposing axes.	(which will measure (1,0,0), (0,1,0) and (0,0,1) and can give an exact zero field powder average)
selectaxis	Allows a mix of LF and TF data, e.g. for a fit. Parameter N=(field axis, 0=X,1=Y,2=Z)+(detector axis, 0=X,3=Y,6=Z). Optionally spin direction x,y,z (default z) then RF direction x,y,z (default y).	selectaxis=8,0,0.1,1 fit3par=selectaxis[0] (fix per spectrum to 2,5,8 to choose detector/group) fit4par=selectaxis[2] (global, allow spin rotation)
goniometer	Pairs of (axis, angle) applied in the order given. Rotates the "sample" with respect to the "instrument". Axes x,y,z rotate around the instrument axes while a,b,c rotate about the (possibly already rotated) crystal axes.	goniometer=z,45
crystalsite	Crystal unit cell point x,y,z. Assumes the spin system has the symmetry of that crystal lattice location, and iterates over all equivalents. Use in conjunction with "tf" or "lf", etc. loopXpar=selectorient splits the different sites into individual rows/columns in the result (no loopXrange line is required)	crystalsite=0.125,0.125,0.125 (silicon bond centre)
bmag(site)	Specify the magnitude of the magnetic field (Tesla). With 3 values, vector field along (nominal field axis, nominal RF B1 axis, 3rd 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", "phasequad", "integral", "fit", "m0", "m1", "m2", "freqspec", "breitrabi"	measure=fit
phasequad	For measure=freqspec,m0,m1,m2: generate signed frequencies according to rotation direction around the field axis.	phasequad=1
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 calculate the value based on bmag: f=A+B*bmag	minfreq=0
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 μs) 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(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
morespin	Additional spin(s) to polarise, or correlate. Spin,x,y,z -> polarise that spin in absolute (crystal) direction. Spin,axis -> polarise along axis (lab) which can be "beam", "field", "detector" or "rf". Spin1,"T",spin2 -> the two spins in a triplet state. Spin1,"S",spin2 -> singlet. "temperature",T sets all spins apart from the muon to Boltzmann populations (e.g. for crystal field use).	morespin=H,beam,e1,T,e2 (H nucleus polarised parallel to initial muon, the two electrons e1 and e2 in a triplet state J=1, m_J=any of -1,0,+1) morespin=temperature,1.4

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 values (starting from 0). Code[i,j] scans values i	loop0par=bmag (simple field scan) loop0par=a(Mu)[1] (varies D) loop0par=lf[0,1]

	and j round the unit circle, where the loop variable is in degrees. Multiple values can be scanned simultaneously, separated with ";"	(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 or in the ALC interface)
loop0range	For measure=integral, the values here are ignored and the data's X axis is used instead. (The line should still be present)	
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. Multiple values to be varied together are separated by ";". A code can be prefixed with "-" to be set to the negative of the fit parameter, or "^" to set to 10 to the power of the parameter (i.e. fitting works with the log of the variable).	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) (varied on log scale)

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.

“PhaseQuad” output mode produces two spectra, for detectors at 0 (as specified) and 90 degrees round the field axis. For a diamagnetic muon and positive bmag, the 90 degree spectrum lags behind the 0 deg one. Loops may not be used.

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. With the additional parameter “phasequad” set, the frequencies are now signed – with positive bmag, a diamagnetic muon will precess with positive frequency while an electron, or triplet muonium, will have negative 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.

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 zero corrections are only strictly correct for times greater than the end of the pulse, and unexpected large values can sometimes be observed within the pulse. For comparison with or fitting

to correctly reduced time dependent data this should not be a problem. Integral counting data on a pulsed instrument should be modelled with “starttime” set appropriately. Asymmetric pulse shapes such as exponential will necessarily introduce phase shifts. Code e0 (exponential with shift) is offset to start at -1 lifetime so the average “arrival time” is zero and the phase is zero for low frequencies, as would be determined experimentally with a TF scan. There are still shifts at higher frequency, close to cutoff, varying as the cube of the frequency or higher. Code “et” is a truncated version which avoids numerical divergence with a fast relaxing component of similar lifetime to the exponential pulse width.

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.

Further fitting details

A set of fit functions QuantumTableDrivenFunction1 – QuantumTableDrivenFunction5 are provided, depending on how many parameters need to be varied. (Mantid does not let user functions vary the number of parameters or their names.) All have additional parameters for scale factor and baseline.

The control table should be called “Tab” – ideally this should be an Attribute to the function but Mantid currently has some bugs regarding Attributes, especially in the interactive fitting dialogs and interfaces.

The Multi Dataset Fitting interface allows several spectra to be fitted simultaneously to the same Quantum function. If these are runs at different fields, the field should be one of the fit parameters (e.g. fit0par=bmag), the parameter (e.g. P0 in the Fit window) should be set to individual, and initialised to the field setpoints. For TF time spectrum data these could be left free, but for LF they should probably be fixed. Other parameters such as hyperfine constants can then be made Global. Similarly, in transverse geometry different detector groups can be fitted, for example fit0par=tf[6,7] and P0 initialised to 0 and 90 for the forward-backward and top-bottom datasets. These phases may need to be left free to correct for exact detector positions and relative efficiencies. Or use fit0par=selectaxis[0] and initialise P0 to 2 and 5 for two perpendicular TF groups and 8 for LF.

Sometimes two parameters may be strongly correlated, such as the muon and proton hyperfine constants when fitting a Δm=0 ALC line, and it is better to fit the sum and difference: see for example fit function QuantumTableDrivenFunction3SD with parameters P1plusP2 and P1minusP2 (and P0).

Do not use any of the "random Monte Carlo" options such as lfrandom when fitting, as the resulting small variations from run to run can upset the fit minimiser.

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.

Stand-alone use

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

It is possible to run QUANTUM on its own using just a Python interpreter and its standard libraries (including numpy). A few options will not be available unless the Mantid libraries can be found – for example crystal fields, crystal space groups and fitting of individual time spectra. For these, QUANTUM will only try to import the libraries on seeing that they are needed.

At the start of your script use:

```
from quantumtabletools import RunModelledSystem
```

Create a dict() corresponding to the contents of the table. Those values which can be a list of one or more numbers must be given as a tuple of numeric values, even if there is only one. Some such as fitfunction and loopXpar must be left as strings.

Then call QUANTUM with:

```
xaxes,yvals,errvals = RunModelledSystem(tableDict)
```

`xaxes` is an array of 1 or 2 axes (numpy arrays).

`yvals` is the result array, generally 2-dimensional though one axis may have size 1.

`errvals` is the same size as `yvals`, and contains errors – generally zero apart from measure=fit.

See "standalone.py" in the examples subdirectory.

Lower level code

It is also 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 gamma 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,rspins,pols)
```

nsites is the number of states, the conversion is from state1 to state2 (integers) at rate nu (inverse microseconds). rspins 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) = Re(sum(ccossin[i] * exp(omega[i]*t))). omega and ccossin are complex arrays, and 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 StaticHam+RFHam0*cos(omegaRF*t)+RFHam90*sin(omegaRF*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 (-omegaRF/2..omegaRF/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,…]