Software Details:

Requirements:
MatLab R2016b or higher
NLSL v 1.5.1+ for Linux, available from
https://www.acert.cornell.edu/index_files/acert_ftp_links.php

Table of Contents:

Installation:

- Extract *CSCA-distro-X.X.tar.gz* to your chosen *CSCA-directory*.
- Confirm that the working *nlsl* executable exists in the *CSCA-directory/CSCA/bin* folder. If not, visit https://www.acert.cornell.edu/index_files/acert_ftp_links.php and download your chosen version of the NLSL package (for most users, this is NLSL.MOMD). Extract the files into the *CSCA-directory/CSCA/bin* folder, and compile the *nlsl* program (see *NLSL Compilation*, below).
- In MatLab, locate *CSCA-directory/CSCA*, right-click it, and choose "Add to Path > Selected folders and subfolders."

NLSL Compilation from Source (hopefully unnecessary):

Note: the currently available distribution of most NLSL programs calls a function that determines CPU time, which is incompatible with many systems and can prevent proper compilation. Furthermore, the current version generates graphics on every fit-iteration, which can slow CSCA and prevent use of other programs.

If you wish to modify NLSL source files to omit references to the CPU-time function, as well as eliminate graphics-generation (strongly recommended), we have included the installNLSL.m file, which modifies the NLSL source to eliminate these references.

To perform this operation, once you have unzipped CSCA-distro-X.X.tar.gz to CSCA-directory,

do the following:

- Download and unzip the NLSL program into *CSCA-directory/CSCA/bin.*
- In Matlab, add *CSCA-directory/CSCA* to your path.
- In the Matlab command line, type:

  install('*/path/CSCA-directory/CSCA/bin*',0)

  for the graphics-free version or

  install('*/path/CSCA-directory/CSCA/bin*',1)

  for the version which displays graphics at each iteration.

Then compile NLSL using the instructions below.

- At the moment, NLSL is only supported as a Linux distribution. First, make sure the appropriate packages are installed. You will need *libx11-dev* and *gcc* (or equivalent fortran and C compilers). To download and install these, simply type:

  >sudo apt-get install libx11-dev
  >sudo apt-get install gcc
  >sudo apt-get update

  and enter your password, when prompted.
- Next, extract NLSL to *CSCA-directory/CSCA/bin* (if you have not done so already) and look for the *makefile* in *CSCA-directory/CSCA/bin/nlsl*. You will need to edit this in accordance with your setup
  - In FFLAGS, you will need to change -m64 to -m32 if using a 32-bit computer.
  - You will need to enable X11 forwarding by setting
    LIB=-L *usr*Xqq/lib64-lX11 -L/usr/X11R6/lib64 (or equivalent for 32-bit architectures)
  - You will need to set your Fortran 77 compiler to *gfortran* (or equivalent)
    F77=gfortran
  - You will need to set your C compiler to *gcc* (or equivalent)
    CC=gcc
- In a terminal, *cd* to the *CSCA-directory/CSCA/bin/nlsl* directory and make sure no '.o' files are present
  >rm *.o
- Compile *nlsl*:
  >make nlsl
- Confirm that the program works by opening the program in the terminal:
  >./nlsl

- Move the new nlsl binary up a directory, to *CSCA-directory/CSCA/bin.*

Usage:

Parameters:

Fit parameters are built in the *params* structure, where users do the following:

- Specify the type of fit to be run:
- Specify how many fits should be run.
- Does the user wish to keep raw data files associated with fitting? (Discarding these drastically reduce computation time).
- What parameters should be explicitly fixed? What are their values?
- What parameters should be explicitly varied?
- What starting value (or range of starting values) should be used for each parameter, if relevant?
- What convergence criteria should NLSL use to evaluate convergence?

The variables used by this macro are defined in Table S1.

Table S1: Variables defined by *cscaPmake* and stored in the 'params' structure.

Variable	Description
params.vary	Names and values of varying float variables. This is a cell-array, with names in the first column, lower-bounds in the second, and upper-bounds in the third.
params.varyGuess	An initial guess at the value of the varying floats. This is a cell-array, with names in the first column and values in the second.
params.fixfl	Names and values of fixed float variables. This is a cell-array, with names in the first column and values in the second.
params.fixint	Names and values of fixed integer values. This is a cell-array, with names in the first column and values in the second.
params.trialNum	Number of fits to be run
params.parType	Type of parameter variation 0 = Single starting point 1 = Monte Carlo sampling 2 = Grid sampling

	3 = Bounded simulated annealing 4 = Bounded particle swarm 5 = Bounded genetic algorithm
params.resType	Type of data resampling (only relevant for params.parType=0-2). 0 = No resampling 1 = Monte Carlo Resampling Note that this requires the params.noise variable to be defined.
params.keepFiles	Should raw files be saved locally? 0 = No (faster) 1 = Yes
params.noise	Noise estimate (for Monte Carlo resampling)

When an analysis is completed, the program will create a new folder containing an *nlslout.mat* structure. If loaded, this will contain the *params* variable. The *params* variable can be re-used, so that users can compute fits from the same random seed of initial guesses. These guesses will be stored in *params.varyGuess*, whose structure will expand to accommodate the initially chosen guesses. Additionally, the software stores the software path (*params.path*) and a marker that indicates that initial guesses are already generated (*params.paramsGenerated).* This marker can be reset to 0 if users want to re-run experiments with new random-guesses, or can be left at 1 if users wish to re-run experiments with the initial random seed.

Running NLSL:

Once a parameter file has been generated, the user should navigate to the data-containing directory. Spectral data should be stored as a two-column file (e.g. *data.dat*) containing B0 values (in Gauss) and spectral intensity values (arbitrary units) separated by commas. Then run:

[*out,clean]=nlsl*(‘data.dat’,params)

This will run NLSL and save the output into a structure called *out*, which contains the variables defined in Table S2. Additionally the function will save *params* and *out* in a directory named after the input filename. An additional structure called *clean* will also be produced and saved. It is identical to *out*, except that it discards fits outside the specified range of best-fit parameters and it does not contain complete chi squared data. The exception: *clean* is not generated for fits without resampling or parameter variation, since only one fit is run in this case.

Table S2: Variables stored in the *out* variable created by *nlsl*.

Variable	Description
out.goodInds	Indices of convergent fits. (Sometimes fits fail to

	converge due to software errors.)
out.fit	Best fit parameters for each run
out.uncertainty	Numerical error-bar produced by NLSL for each fit. Not a good estimate of true uncertainty, since it is based on curvature near the optimum.
out.spec	Spectrum value produced by NLSL
out.specuncertainty	Spectrum uncertainty produced by NLSL
out.B0	The B0 value from NLSL
out.B0end	The B0eff value from NLSL
out.chiSq	The χ^2 value from NLSL
out.redChiSq	The reduced χ^2 value from NLSL
out.fitVarName	The names of fitting variables (in order – note that these may be rearranged from their order in the 'params' file).
out.Bfield	Processed magnetic field data for each fit, concatenated
out.Idata	Processed spectral data at each fit, concatenated
out.Ifit	Fitted spectral data for each fit, concatenated
out.limits	The constraint interval for each parameter, initially specified by *params.vary*
out.intData	All χ^2 values collected by software, regardless of status as a local minimum.
out.intVarNames	Names of each column in the *out.intData* array.

In general, *clean* is preferable to *out* for analysis purposes, since *out* will contain some local minima outside of user-defined constraints. However, *out* is necessary for plotting the χ^2 landscape.

Functions for further analysis of data:

To evaluate outputs, CSCA includes a few functions designed for data visualization, which work by mass-producing figures detailing parameter distributions for easy consideration. These functions are presented in Table S3.

Table S3: Functions for mass-producing figures for fit analysis. The variable *mult* is the maximum admissible value of the ratio of reduced $\chi^2/\ \chi^2_{min}$ for a fit to appear in an analysis. Appropriate values for *mult* depend on the specifics of the data which are collected, since (by construction) data with very small χ^2 minima will need substantially greater values of *mult* to include similar numbers of local minima.

Function	Description	Usage
chSqPlot	Plot reduced χ^2 vs. every pair of fit parameters. Only valid for	chSqPlot(out,*clean*, mult,showLocs,figNum)

	multi-parameter fits. Requires *out* and *clean*. Displays the global best fit (green circle) as well as the geometric median (yellow square), the medoid (yellow circle), the marginal median (yellow diamond), and the mean (yellow triangle). Also displays all local minima used for these calculations (in purple). The input *showLocs* gives users the option to 0 = hide explicit minima 1 = show local minima (purple); omit global minimum. 2 = Show local minima and global minimum. *figNum* specifies a figure number where the plots should appear.	
clusterPosition	For a given fit, give numerical estimates for the best-fit of each parameter. Inputs are *clean*, *mult*, *confInt* – a statement of the size of the confidence interval that interests you, between 0 and 100%, and *limits* (you should use *clean.limits* unless you manually want to adjust the domain of your cluster). Outputs are the cluster's mean position, *meanVal*, its marginal median, *margMed*, its geometric median, *geoMed*, its medoid, *medoid*, and a set of intervals for each parameter, which describe the confidence interval produced from a univariate histogram of the	[meanVal,margMed,geoMed,medoid,interval]=clusterPosition(clean,mult,confInt,limits)

	data. **Coordinates are specified as a series of values that correspond directly to the variables named in *clean.fitVarNames*.**	

Recommended approach for fitting

CSCA implements several methods for global optimization. Some of these, such as particle swarm optimization (PSO) and genetic algorithms (GA) are popularly implemented in other EPR analysis software. Others, such as Monte Carlo Levenberg-Marquardt (MCLM) and simulated annealing (SA) are uniquely implemented in this tool.

In general, the relative speed of each method depends on the dimensionality of the parameter space, as well as the objective (modeling) function, and the constraints imposed on the problem. Therefore, no specific method is guaranteed to be the fastest or the slowest under various conditions.

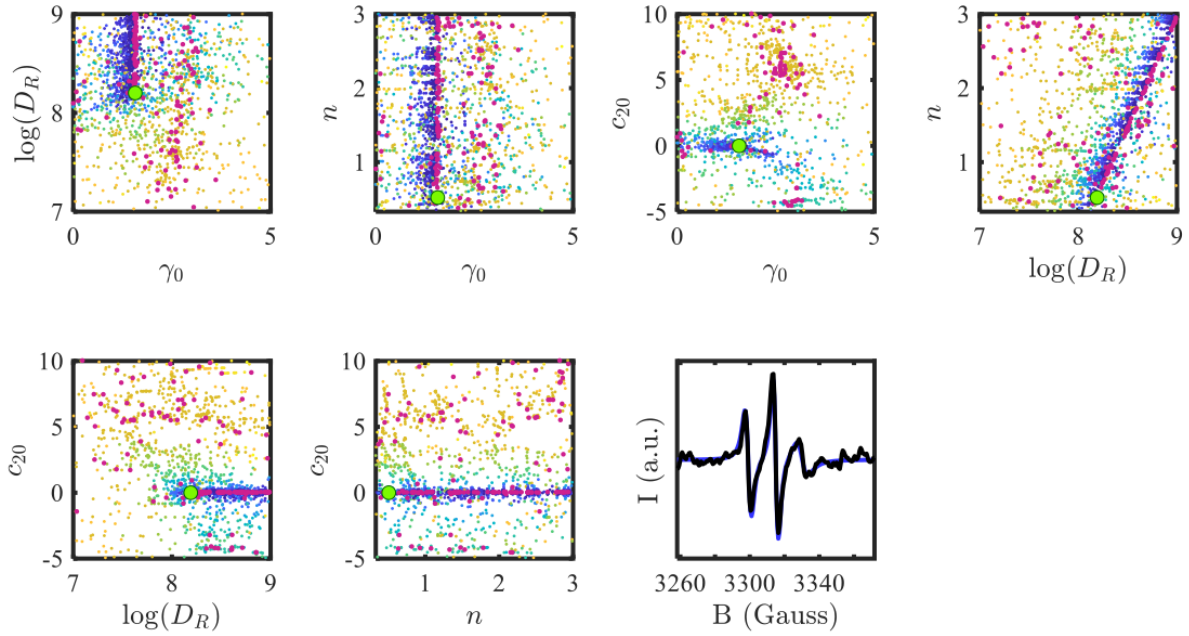
However, in our experience fitting EPR spectra, it is abundantly clear that MCLM is the most efficient way to conduct the tests. Since most of these tasks are stochastic, the time elapsed during each fit is variable – but a simple survey of the time required to fit *test1234.dat* by each method is included below:

- Monte Carlo Levenberg Marquardt (500 iterations)
  - Found global optimum? (Yes)
  - Time elapsed (144s)
- Simulated annealing (SA)
  - Found global optimum? (Yes, on second attempt. First attempt got caught in a local minimum.)
  - Time elapsed (317s – first run;
- Particle swarm optimization (PSO)
  - Found global optimum? (Yes)
  - Time elapsed (922s)
- Genetic algorithm (GA)
  - Found global optimum? (Yes)
  - Time elapsed (1401s)

Thus, while PSO and GA are more generally robust algorithms for nonlinear fitting, given the small size of our problem, MCLM is much faster (and in our experience has always reached the same conclusions). Moreover, since MCLM detects so many more local minima, it is the only approach for which cluster-analysis provides suitable descriptions of fit parameters. **We therefore recommend an MCLM approach for identifying the global fit, and for accurately describing the values of fit parameters.**

Avoiding oversfitting:

In our experience, the type of large, many-parameter fits which would necessitate more complex global optimization functions are not advisable, due to the risk of overfitting. Below is a precise example of what we mean:



In this example, a spectrum was fit assuming a degree of anisotropy (n) in the rotational diffusion parameter. This resulted in an elongated, linear cluster of local minima – a strong indication of overfitting, and a serious problem for someone trying to make meaningful statements about rotational diffusion.

In general, we recommend avoiding the use of more than 3-4 variables in your model – keep everything else fixed. In general, overfitting occurs because it is impossible to distinguish between distinct models which could give rise to the same behavior. Therefore, keeping your models simple will reduce the possibility of this occurrence.

Tutorial: Running a basic fit.

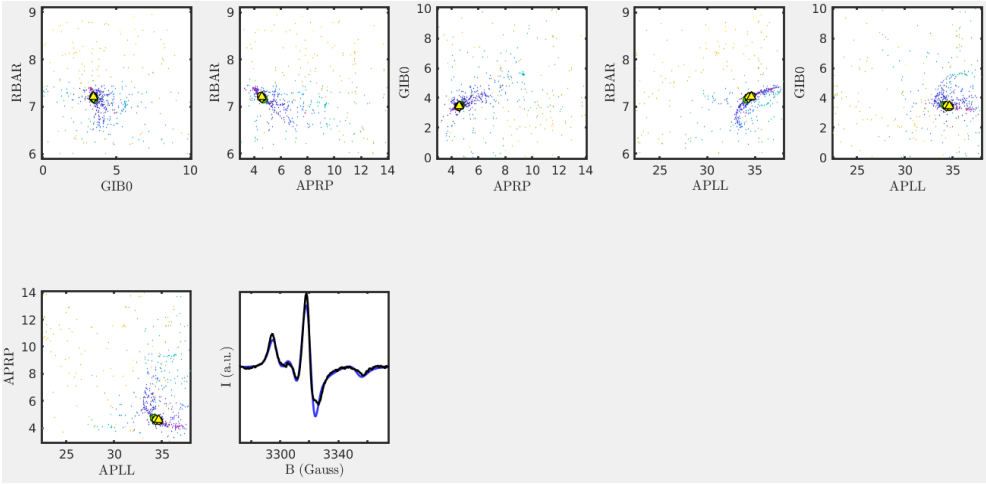
Included with the installation is a folder called *test*, which contains a sample data-set for analysis, *test1234.dat*. Navigate to *test* and open the file *cscaPmake*. This file, once run, will create a *params* variable indicating that the data should be fit 100x (*params.trialNum=100*) in order to determine appropriate values for *rbar,* the values *aprp* and *apll* of the hyperfine tensor, and *gib0*, the Gaussian line-broadening.

Other common parameters to vary are commented-out of the code, but can be added in by modifying the *params.vary* object. Just make sure that there is no overlap between variables in *params.fixfl* and *params.vary*. For MOMD fits, you must include the *c20* parameter as a fit variable.

In order to analyze *test1234.dat*, open MatLab and follow the protocol below:

1. Double-check that *CSCA-directory/CSCA* and all its subdirectories exist on your path (if not, right-click and select 'add to path'.
2. In the *test* folder, run the command *cscaPmake* to produce the *params* structure.
3. Type *[out,clean]=nlsl('test1234.dat',params)* into the matlab command terminal, and wait until analysis is complete. Once analysis is complete, confirm that in the newly-created *test/test1234* folder, the file *nlslout.mat* has been generated. You can load this if you need to clear your variables; it will reproduce *params*, *out*, and *clean*.
4. Plot the reduced χ^2 landscape by typing *chSqPlot(out,clean,2.5,2,13)*. This should generate a plot of the reduced χ^2 as it depends on each parameter-pair, Local minima with *mult*<2.5 will be plotted, along with typical cluster positions. The figure should appear on Figure 13, as was specified in the inputs.
5. Quantify the position of each measure of central tendency by typing *[meanVal,margMed,geoMed,medoid,interval]=clusterPosition(clean,2.5,50,clean.limits)*. This will report each measure of central tendency, as well as bounds on the 50% confidence interval, for all local minima with *mult*<2.5. Note that variables are reported in the order listed in *clean.fitVarNames*.

For an example of how these commands can be combined in a simple script, refer to *exampleScript.m* in the same directory. The plot will look something like this:



Tutorial: Analyzing multiple fits by MOMD

In this tutorial, we will employ the same approach but fit multiple spectra in succession. This file, once run, will create a *params* variable indicating that the data should be fit 100x (*params.trialNum=100*) in order to determine appropriate values for *rbar,* the *c20* ordering parameter, and *gib0*, the Gaussian line-broadening.

Note that this analysis uses an MOMD fit that depends on 10 orientations (*nort=10*) in order to assign a value to *c20*. **This will slow the analysis, which will take approximately 10-15 minutes (in total for the three spectra) on a typical machine.**

1. Double-check that *CSCA-directory/CSCA* and all its subdirectories exist on your path (if not, right-click and select 'add to path'.
2. Navigate to the *CSCA-directory/test* folder, and in MatLab open a new script.
3. Open *cscaPmake2* to see the parameters you are fitting for – *rbar, c20,* and *gib0*.
4. In your script, define a cell called *filenames*, which contains the names of the files you want to analyze:

```
filenames={'test2345.dat','test3456.dat','test4567.dat'};
```

5. Below this code, write a for-loop to make the *params* structure and analyze each file

```
for i=1:length(filenames)
        % Generate Parameters
        cscaPmake2;
        % Run CSCA
        [out,clean]=nlsl(filenames{i},params);
end
```

6. Finally, write another for-loop to display the data. (Unfortunately, this cannot be included in the previous loop because these programs close graphics windows.)

```
for i=1:length(filenames)
        % navigate to directories where output files are saved
        cd(filenames{i}(1:end-4));

        % load data
         load('nlslout.mat');

        % navigate back to test directory
        cd ..
```

```
        % Plot
        chSqPlot(out,clean,2.5,2,i)
end
```

For an example of this complete script, refer to *exampleScript2.m* in the same directory.

Common Problems

- If you get an error message saying that the software is unable to find a log file, your *nlsl* executable likely isn't working. Try navigating to *CSCA-directory/CSCA/bin* in terminal, and running the command

  ./nlsl

  You may need to grant permissions to the binary file. If this does not work, try recompiling NLSL for your Linux system.

Float variables that can be fit or declared in NLSL

The following are taken from documentation on NLSL software – they can all be specified as fixed values via params.fixfl or varied using params.varyGuess and params.vary. The most important variables to specify/analyze are listed in bold.

>PHASE|Phase of spectrum (dispersion=90)

>WINT0|Isotropic inhomogeneous linewidth
>WINT2|Psi-dependent inhomogeneous linewidth

>**GXX**|x-component of g-tensor (cartesian)
>**GYY**|y-component of g-tensor (cartesian)
>**GZZ**|z-component of g-tensor (cartesian)
>G1|0,0 (isotropic) spherical component of g-tensor
>G2|2,0 (axial) spherical component of g-tensor
>G3|2,2 (rhombic) component of g-tensor
>GPRP|perpendicular component of axial g-tensor
>GRHM|rhombic distortion of axial g-tensor
>GPLL|parallel component of axial g-tensor

>AXX|x-component of g-tensor (cartesian)
>AYY|y-component of g-tensor (cartesian)

>AZZ|z-component of g-tensor (cartesian)
>A1|0,0 (isotropic) spherical component of g-tensor
>A2|2,0 (axial) spherical component of g-tensor
>A3|2,2 (rhombic) component of g-tensor
>**APRP**|perpendicular component of axial g-tensor
>ARHM|rhombic distortion of axial g-tensor
>**APLL**|parallel component of axial g-tensor

Note that R values specify the base-10 log of the property described. Anisotropy values do not.

>RX|x-component of rotational diffusion tensor
>RY|y-component of rotational diffusion tensor
>RZ|z-component of rotational diffusion tensor
>**RBAR**|average rot'l diffusion rate = (Rx Ry Rz)^1/3
>N|Rotional anisotropy = 2*Rz/(Rx+Ry)
>NXY|Ratio of Rx/Ry
>RPRP|Perpendicular component of axial diffusion tensor
>RRHM|Rhombic distortion of axial diffusion = Rx/Ry
>RPLL|Perpendicular component of axial diffusion tensor

>PML|Model-dependent diffusion parameter: z rotation
>PMXY|Model-dependent diffusion parameter: xy rotation
>PMZZ|Model-dependent diffusion parameter: internal z rot'n

>DJF|Rate of jumping among <ist>-symmetric sites
>DJFPRP|Perpendicular component for anisotropic viscosity model

>OSS|Heisenberg spin-exchange rate

>PSI|Director tilt angle

>ALPHAD|Alpha Euler angle for diffusion tilt
>BETAD|Beta Euler angle for diffusion tilt
>GAMMAD|gamma Euler angle for diffusion tilt

>ALPHAM|Alpha Euler angle for magnetic tilt
>BETAM|Beta Euler angle for magnetic tilt
>GAMMAM|Gamma Euler angle for magnetic tilt

>**C20**|Y(2,0) coefficient of orienting potential
>C22|Y(2,2) coefficient of orienting potential

>C40|Y(4,0) coefficient of orienting potential
>C42|Y(4,2) coefficient of orienting potential
>C44|Y(4,4) coefficient of orienting potential

Integer variables that can be fit or declared in NLSL

We strongly recommend using the values selected in the tutorials

>IN2|Twice the nuclear spin

>IPDF|Diffusion model (0=Brown,1=non-Brown,2=aniso.visc.)

>IST|Number of symmetry-related sites for discrete jump

>ML|0/1 for jump/free diffusion around molecular z
>MXY|0/1 for jump/free diffusion around molecular x,y
>MZZ|0/1 for jump/free INTERNAL diffusion around molecular z

>LEMX|Maximum index for even L-values in basis
>LOMX|Maximum index for odd L-values in basis

>KMX|Maximum index for K-values in basis
>KMN|Minimum index for K-values in basis

>MMX|Maximum index for M-values in basis
>MMN|Minimum index for M-values in basis

>IPNMX|Maximum index for pI values in basis

>NSTEP|Number of steps in CG matrix solution