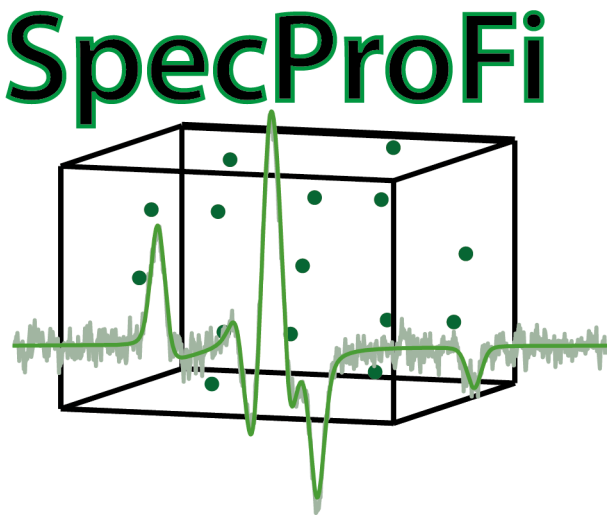
SpecProFi User Manual

Version 1.0.1



by

Stephan Rein

stephan.rein@physchem.uni-freiburg.de

https://www.radicals.uni-freiburg.de/de/software

Contents

Preface

SpecProFi has been developed in the group of Prof. Dr. Stefan Weber at the University of Freiburg, Freiburg im Breisgau, Germany, during the last couple of years as a software for advanced data processing and analysis of EPR data. This manual describes the general usage of SpecProFi and gives an overview of the maths behind that has been implemented. If you use SpecProFi for your own research and publish results accordingly, please give credits citing the appropriate reference :

S. Rein, Development of advanced simulation and analysis programs for EPR spectroscopy, dissertation, Freiburg, **2019**.

Some parts of this Manual were taken form the dissertation [1] of which the author is the author of this manual.

A number of people have helped shaping SpecProFi and the ideas behind. First and foremost, Prof. Dr. Stefan Weber and Dr. Sylwia Kacprzak (now Bruker Biospin).

Freiburg, October 2019

Stephan Rein

Disclaimer

SpecProFi is provided "as is", without warranty of any kind, express or implied, including but not limited to the warranties of merchantability, fitness for a particular purpose and noninfringement.

1 Introduction

1.1 General Description

SpecProFi is a MATLAB-based EPR data processing and fitting framework, supporting global analysis. SpecProFi relies on the simulation functions of the EasySpin simulation toolbox [2, 3]. The EPR data processing framework is a collection of individually callable functions for processing raw data, including phase correction, baseline correction and data denoising. The analysis/fitting framework consists of one user-callable function, that accepts various user-defined settings. This function internally invokes the simulation routines of the well-established and comprehensive simulation framework EasySpin and supports different experimental data sets to be analyzed globally. This includes the simultaneous analysis of combinations of cw-EPR data measured at different microwave frequencies, different harmonics, ENDOR and cw-EPR data, or cw-EPR data recorded under solid-state and liquid-state conditions. There are no restrictions of SpecProFi as long as there is an EasySpin simulation routine available for the simulation of a certain EPR experiment.

1.2 Installation Instructions

SpecProFi is available free of charge and open source. It can be downloaded from our homepage `https://www.radicals.uni-freiburg.de/de/software/specprofi` It is sufficient to download the SpecProFi folder as well as the current EasySpin version `http://www.easyspin.org/` and set the path to your MATLAB workspace.

2 Data processing

2.1 List of all data processing functions

Alphabetically ordered processing functions of SpecProFi with cross reference:

1. `APC()`
2. `baseline_correction()`
3. `diff_Tikhonov()`
4. `diff_Tikhonov_F()`
5. `discrete_wt()`
6. `Gaussian_smooth()`
7. `Hilbert_transform()`
8. `idiscrete_wt()`
9. `int_and_diff()`
10. `LoadEPRfile()`
11. `normalize()`
12. `phase_offset()`
13. `pseudomod()`
14. `wavelet_denoising()`
15. `SVD_denoising()`

2.2 Loading data files

For loading some experimental EPR file, the `LoadEPRfile()` function can be used. Either text files or Bruker files can be loaded:

Script 1: Loading experimental EPR data sets.

```
% Load a text file
[field, spc] = LoadEPRfile('Sample123.txt');

% Load a Bruker file (.DSC and .DTA or .par and .spc)
[field, spc] = LoadEPRfile('Sample123');
```

The `LoadEPRfile()` automatically determines if the signal has only a real part or a real and imaginary part. In case the loaded signal was recorded using a quadrature detection, the signal (`S`) is composed of a real and imaginary part:

```
[B, S] = LoadEPRfile('Some_ELEXSYS_file')
```

with

$$S = S_{\text{real}} + \mathrm{i} \cdot S_{\text{imag}}$$

Two additional strings can be passed to the function to integrate the signal or reconstruct the imaginary part (the function only uses the reconstruction algorithm if no imaginary part was recorded).

Script 2: Additional options for a real valued data set.

```
% Load a Bruker file and reconstruct the imaginary part automatically
[field, spc] = LoadEPRfile('Sample123', 'imag');

% Load a file and return the integrated signal
[field, spc] = LoadEPRfile('Sample123', 'integrate');

% Load a file and return the integrated signal with imaginary part.
% The imaginary part is reconstructed if not present in the loaded data set
[field, spc] = LoadEPRfile('Sample123', 'imag', 'integrate');
```

If you want to have more detailed information about the function or about the input and output parameter then use the help function of MATLAB.

```
>> help LoadEPRfile
```

The output of the help function for the `LoadEPRfile` function is shown in Script 3.

Script 3: Information shown on the console when invoking the help function

```
Usage
-----
Load EPR files by providing either text file formats or
Bruker file formats (EMX, ESP, ELEXSYS)

INPUT:
  data_file               - string
                            Filename of the experimental data set

OUTPUT:
  xaxis                   - vector
                            X-axis data vector

  signal                  - vector
                            Signal data vector

Notes
-----
EPR data import function. The function can read text file formats,
provided as .txt or as .dat. Two columns (field, intensity) are expected
for a signal with only real entries. Three columns (field, realpart,
imaginarypart) are expected for a signal which was recorded with
quadrature detection.
Furthermore, Bruker file formats can be read. For this, both files, the
binary (.DTA or .spc) and the description file (.DSC or .par) needs to be
present. If a Bruker file should be read the filename need to be given
without file format.

Examples
--------
[B, spc] = LoadEPRfile('Test_data.txt')

[B2, spc2] = LoadEPRfile('Some_EMX_file')


Copyright (c) 2019, Stephan Rein
```

2.3 Normalization

Normalization of the experimental signal can be carried out using the `normalize()` function of SpecProFi. Different approaches for a normalization are available. The default method is the normalization to the absolute maximum of the signal. Alternatively a normalization to the (non-absolute) maximum of the signal, to the area, or to the area normalized to the number of data points can be carried out.

Script 4: Normalization function of SpecProFi.

```
% Use the default normalization to the absolut maximum
signal = normalize(signal);

% Normalization to the (non-absolute maximum)
signal = normalize(signal, 'max');

% Normalization to the area of the signal
signal = normalize(signal, 'area');

% Normalization to the area of the signal normalized to the
% number of data points
signal = normalize(signal, 'area_per_point');
```

Comparison between normalization of a signal to its area and to its area with renormalization to the number of points is shown for two different signals with a different number of points in Figure 1.

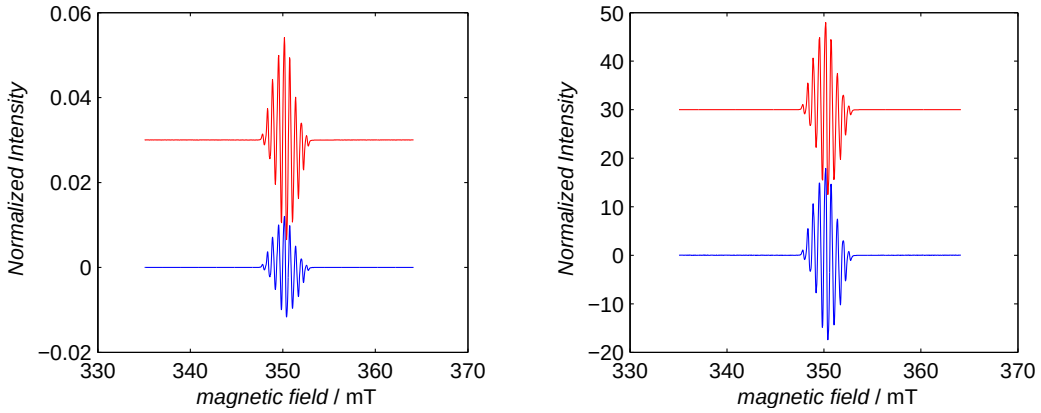


Figure 1: (Left) Two signals with a different number of points normalized to their area. One signal was shifted vertically for the sake of clarity. (Right) Two signals with a different number of points normalized to their area with renormalization to their number of points. One signal was shifted vertically for the sake of clarity.

2.4 Baseline correction

A baseline correction can be carried out by fitting a polynomial function to a specified signal range (at the low- and high-field edge) by invoking the function `baseline_correction()`. The default setting is a polynomial of degree one and 10% of the signal of the low-field and high-field edge for determining the polynomial function. The polynomial degree and spectral region can be set by the user if required.

Script 5: Baaseline corrections of some EPR spectrum.

```
% Using default setting (10% and polynomial degree = 1)
signal = baseline_correction(signal)

% Using a third order polynomial function
signal = baseline_correction(signal, 3)

% Use a third order polynomial function, 15 percent of the spectrum for
% determination and additionally return the polynomial function
[signal, polyn] = baseline_correction(signal, 3, 15)
```

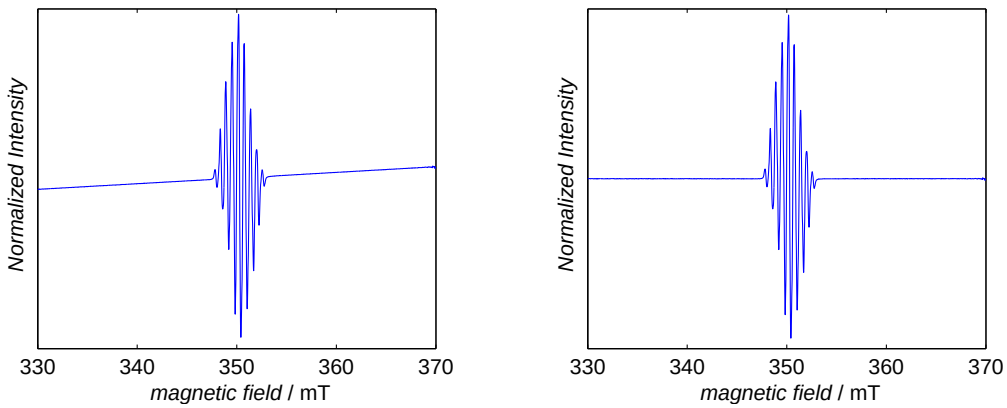


Figure 2: Example for a baseline correction, using the default settings of `baseline_correction()`.

2.5 Reconstruction of the imaginary part (Hilbert transform)

To reconstruct the imaginary part, if not already available in the raw data, a Hilbert transform is applied to the real-valued data set using the function `Hilbert_transform()`. The implemented algorithm is according to [**?**].

Script 6: Baaseline corrections of some EPR spectrum.

```
% The Hilbert transform function reconstructs the imaginary part
[realpart,imagpart] = Hilbert_transform(signal);
```

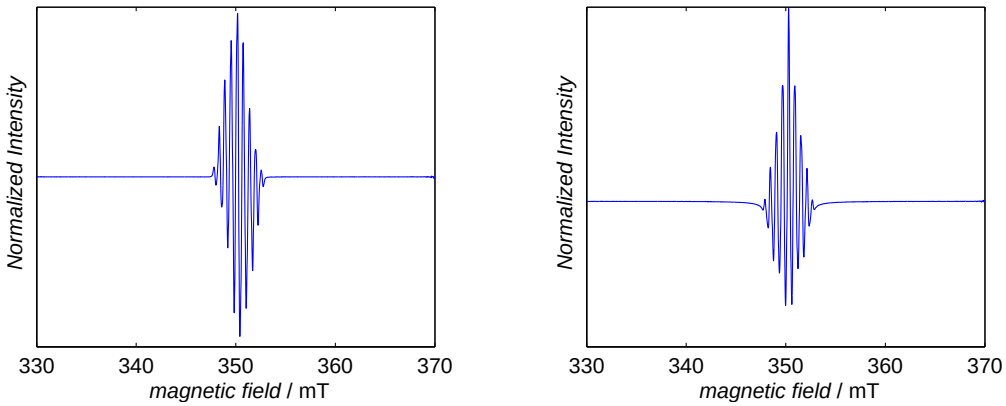


Figure 3: (Left) Real-valued EPR signal. (Right) Reconstructed imaginary part using the `Hilbert_transform()` function.

2.6 Phase correction

The phase of the signal can be corrected manually with a user defined phase angle γ by applying the `phase_offset()` function. Alternatively, the phase can be automatically corrected using an algorithm implemented in the `APC()` function of SpecProFi. Both functions use the imaginary part if provided. Otherwise, the imaginary part is automatically reconstructed using a Hilbert transform.

Phase offset

Examples for the phase offset function is shown in Script 8 and Figure 6.

Script 7: Baaseline corrections of some EPR spectrum.

```
% Offset by an angle of phi = pi/5
phi = pi/5
signal = phase_offset(signal, phi)

% Providing realpart and imaginary part separately
phi = pi/5
signal = phase_offset(realpart, imagpart, phi)

% Getting the realpart and imaginary part separately back
phi = pi/5
[realpart, imagpart] = phase_offset(signal, phi)
```

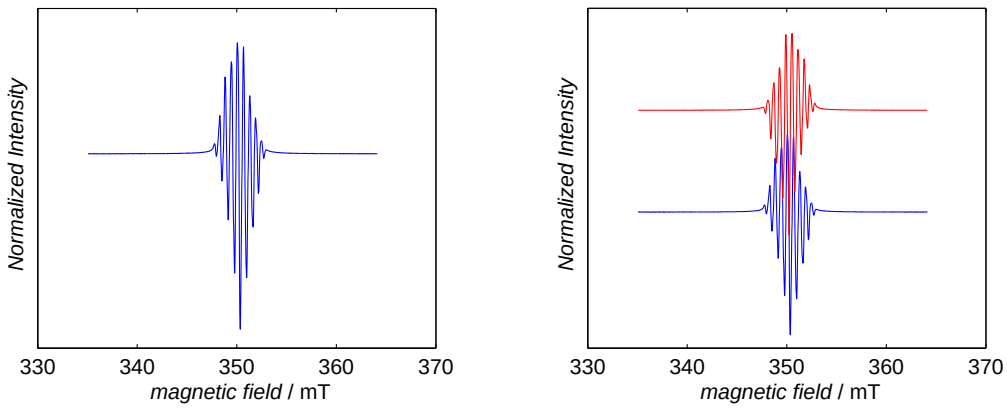


Figure 4: (Left) Real part of a signal with a phase offset of $\phi = \pi/3$. (Right) Real part (blue) and imaginary part (red) of a signal with a phase offset of $\phi = \pi/3$. The imaginary part was shifted for clarity.

Automatic phase correction

Examples for the phase offset function is shown in Script 8 and Figure 6.

Script 8: Baseline corrections of some EPR spectrum.

```
% Standard automatic phase correction
signal = APC(signal);

% Phase correction of a absorptive signal
signal = APC(signal, 'Harmonic', 0);

% Getting the the phase corrected signal and the
% phase angle phi back
[signal, phi] = APC(signal);
```

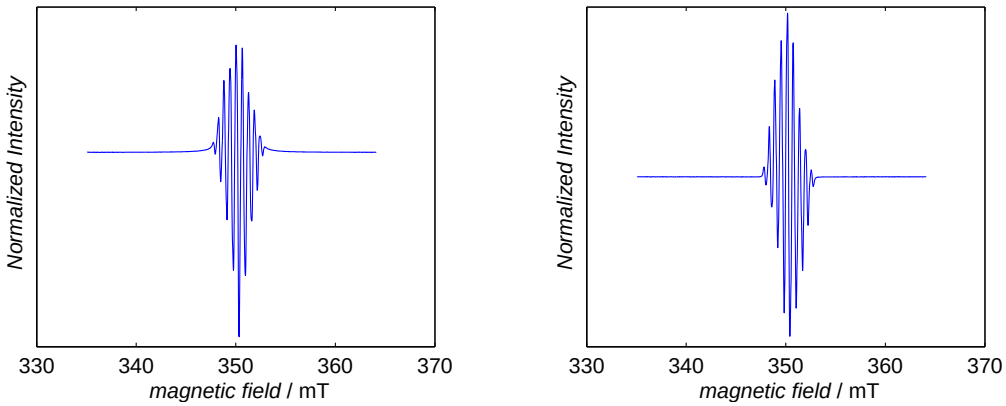


Figure 5: (Left) Signal, recorded with a bad phase. (Right) Signal after using the `APC()` function of SpecProFi.

2.7 Integration or differentiation

The signal can be integrated or differentiated using the `int_and_diff()` function. For integration a cumulative summation algorithm is used. For differentiation the function uses a midpoint differential matrix approach. This matrix is ill-conditioned and for the differentiation of not noise-free experimental data the `pseudomod,diff_Tikhonov()` or `diff_Tikhonov_F()` is recommended.

Script 9: Integration and differentiation of signals.

```
% Integration of a signal
signal = int_and_diff(signal, 1);

% Doubel integration of a signal
signal = int_and_diff(signal, 2);

% Differentiation of a signal
signal = int_and_diff(signal, -1);
```

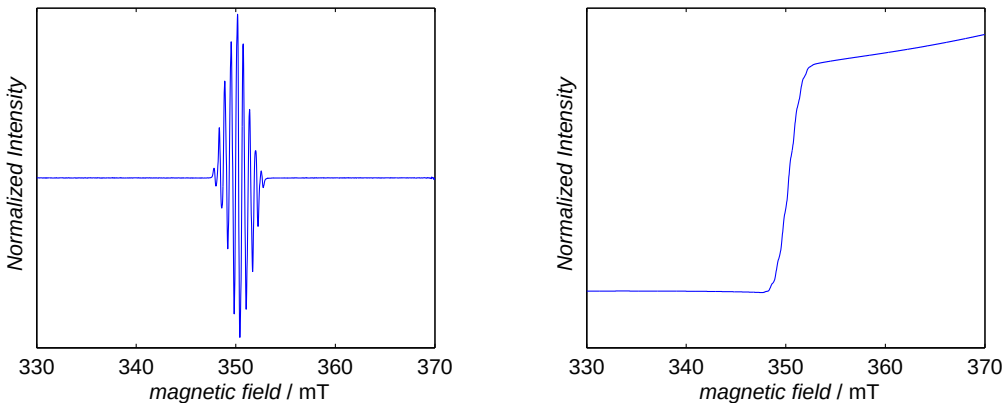


Figure 6: (Left) Original signal before integration. (Right) Signal after double integration using the `int_and_diff()` function of SpecProFi.

2.8 Pseudo-field modulation and Tikhonov regularization for differentiation

To differentiate a signal, different methods are implemented to stabilize the differentiation procedure, which is an ill-posed problem. Pseudo-field modulation [**?**] uses a convolution of the signal with the Fourier transform of a Bessel function of first kind to render the procedure better-conditioned. Alternatively Tikhonov regularization can be used to stabilize the differentiation.

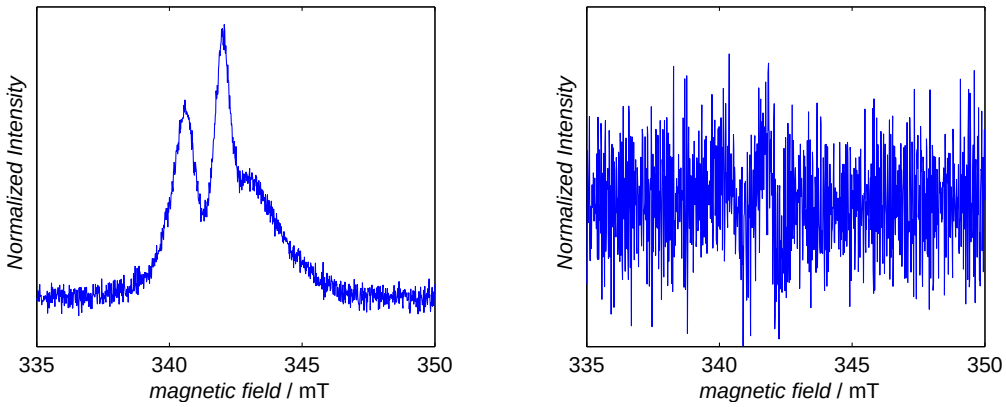


Figure 7: (Left) Original noisy signal before differentiation. (Right) Signal after numerical differentiation using the `int_and_diff()` function of SpecProFi.

Pseudo-field modulation

Pseudo-field modulation can be carried out using the `pseuodmod()` function of SpecProFi. To apply a pseudo-field modulation, a modulation amplitude needs to be provided, as well as the magnetic field vector (in mT). Examples for pseudo-field modulations are shown in Figure 8.

Script 10: Pseudo-field modulation function of SpecProFi.

```
% Pseudo-field modulation with a modulation
% amplitude of 0.1 mT
signal = pseudomod(B, signal, 0.1);

% Pseudo-field modulation with a modulation
% amplitude of 0.25 mT
modamp = 0.25
signal = pseudomod(B, signal, modamp);

% Pseudo-field modulation with a modulation
% amplitude of 0.1 mT. Additionally the integrated
% (denoised) signal is retuned
[signal, signal_denoised] = pseudomod(B, signal, 0.1);

% Pseudo-field modulation with a modulation
% amplitude of 0.1 mT using additional truncation
% in the Fourier space after 100 points
cutoff = 100;
signal = pseudomod(B, signal, 0.1, cutoff);
```

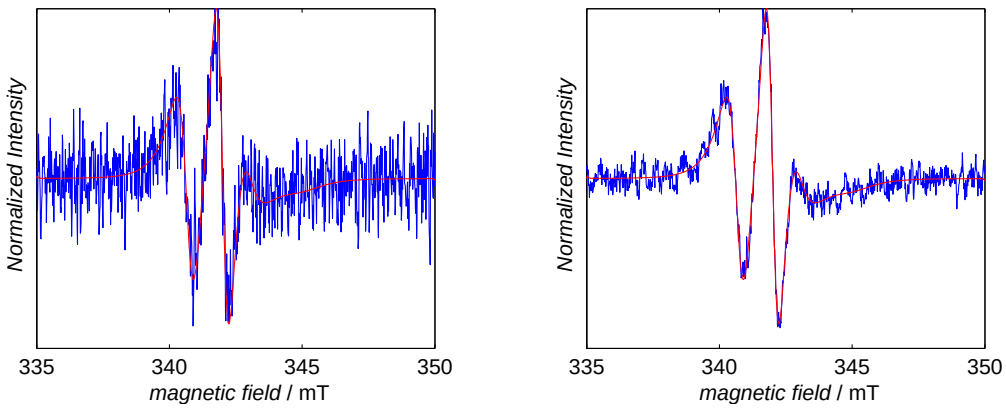


Figure 8: (Left) Using pseudo modulation with 0.1 mT modulation amplitude. (Right) Using pseudo modulation with 0.25 mT modulation amplitude. The known "optimal" solution is illustrated in red.

Tikhonov regularization for differentiation

Two functions, `diff_Tikhonov()` and `diff_Tikhonov_F()`, are available in SpecProFifor differentiation of a EPR signal using Tikhonov regularization. `diff_Tikhonov()` uses a combination of a second derivate operator multiplied with a first derivative operator as penalty matrix while `diff_Tikhonov_F()` carries out a regularization in Fourier space using a unit matrix as penalty expression. Additional options can be provided to both methods, submitted in a structure. Examples for both methods are shown in Figure 10.

Script 11: Tikhonov regularization methods available in SpecProFi.

```
% Standard Tikhonov regularization in field domain
signal = diff_Tikhonov(signal)

% Standard Tikhonov regularization in Fourier space
signal = diff_Tikhonov_F(signal)

% Passing Options (for detailed information use the help function)
Opt.method = 'AIC'   % Using AIC method for determining alpha
Opt.eval = 'fast'    % Faster evaluation using interpolative methods
Opt.adjustment = 'small'; % Reducing alpha by a factor of 10
Opt.bound = 'off';   % Disables cyclic boundary conditions
signal = diff_Tikhonov(signal, Opt)
```

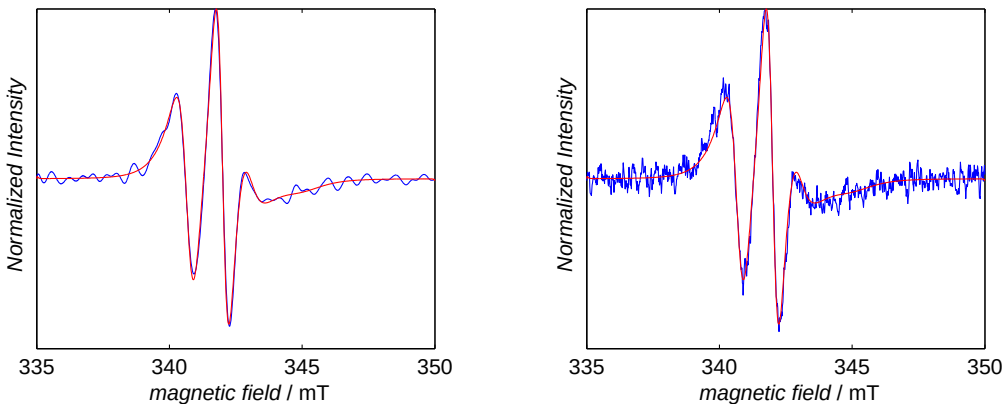


Figure 9: (Left) Tikhonov regularization of the signal using the `diff_Tikhonov()` function of SpecProFi. (Right) Tikhonov regularization of the signal using the `diff_Tikhonov_F()` function of SpecProFi. The known "optimal" solution is illustrated in red.

2.9 Denoising methods

SpecProFi provides several denoising methods which are listed in the following.

Wavelet transform denoising and discrete wavelet transform

SpecProFi provides a Wavelet transform denoising tool, implemented in the `wavelet_denoising()` function. Different wavelet families are available, the default is the Coiflets-4 wavelet.

Script 12: Denoising using the

```
% Standard Wavelet denoising in SpecProFi
signal = wavelet_denoising(signal)

% Using a specified wavelet family
family = 3;
signal = wavelet_denoising(signal, family)

% Using a specified wavelet family and maximum decomposition
% level
family = 2;
maxlevel = 4;
signal = wavelet_denoising(signal, family, maxlevel)

% List of implemented Wavelet families
%   Number                family
%     1               Daubechies D6-Wavelet
%     2               Daubechies D10-Wavelet
%     3               Daubechies D20-Wavelet
%     4               Symlets 10-Wavelet
%     5               Coiflets 4-Wavelet
%     6               Haar-Wavelet
```

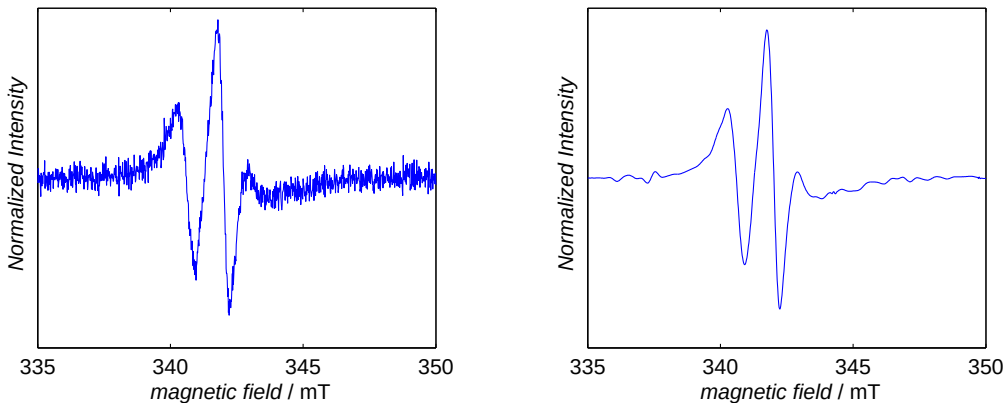


Figure 10: (Left) Noisy signal. (Right) Denoised signal using the default settings of the `wavelet_denoising()` of SpecProFi.

Discrete wavelet transform (DWT) and inverse discrete wavelet transform (DIWT) can be performed using the `discrete_wt()` or the `idiscrete_wt()` function of SpecProFi.

Script 13: Discrete wavelet transform and inverse wavelet transform, implemented in SpecProFi.

```
% Standard discrete wavelet transform of SpecProFi
[trend, detail] = discrete_wt(signal)

% Discrete wavelet transform with a user-specified wavelet family
family = 4;
[trend, detail] = discrete_wt(signal, family)

% Standard inverese discrete wavelet transform of SpecProFi
signal = idiscrete_wt(trend, detail)

% Discrete inverse wavelet transform with a user-specified wavelet family
family = 4;
signal = idiscrete_wt(trend, detail, family)
```

An example for a discrete wavelet transform is shown in Figure 11.

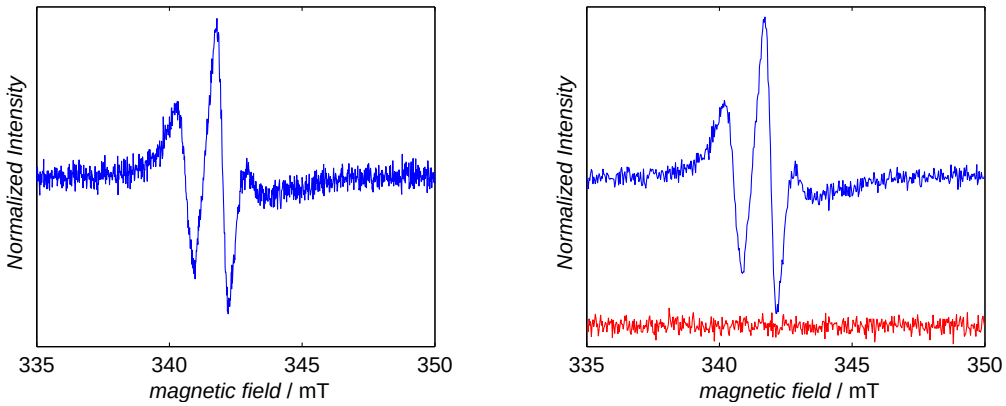


Figure 11: (Left) Noisy signal. (Right) Signal after the discrete wavelet transform. The trend is illustrated in blue while the detail part is illustrated in red.

Gaussian smoothing

Gaussian smoothing, implemented in the `Gaussian_smooth()` function, is a convolution of the signal with a Gaussian function. This leads to a smoothing of the signal. The magnetic field needs to be provided as well as the full width at half maximum (FWHM) of the Gaussian function in mT.

Script 14: Gaussian smoothing of EPR signals

```
% Gaussian smoothing using a FWHM of 0.3 mT
FWHM = 0.2 % Full width at half maximum in mT
signal = Gaussian_smooth(B, signal, FWHM)

% Using a broad Gaussian for smoothing
FWHM = 0.5 % Full width at half maximum in mT
signal = Gaussian_smooth(B, signal, FWHM)
```

An example for a Gaussian smoothing is shown in Figure 15.

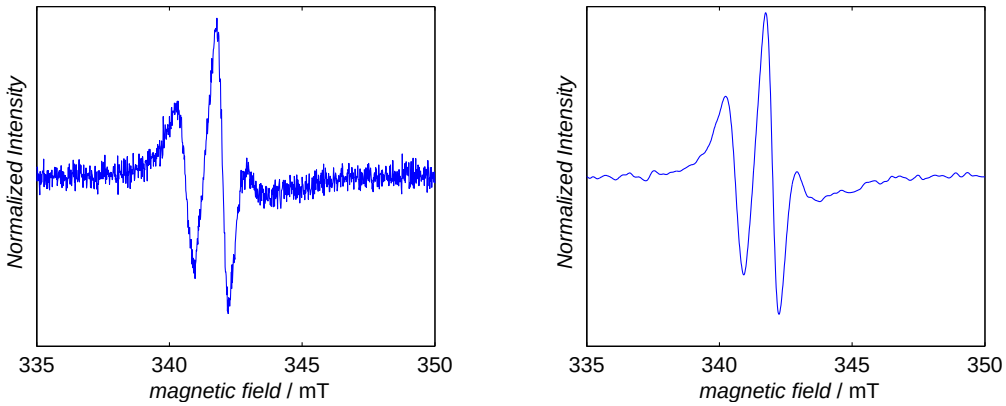


Figure 12: (Left) Noisy signal. (Right) Signal after Gaussian smoothing using the `Gaussian_smooth()` of SpecProFi with a FWHM of 0.2 mT.

SVD-based denoising

SpecProFiprovides the function `SVD_denoising()` for the singular-value-decomposition-based denoising of a signal. The EPR signal is periodically arranged in a Hankel-type matrix. Subsequently, a singular value decomposition of this matrix is carried out and singular values below a certain threshold are set to zero. Per default, the threshold is determined algorithmically. Denoising using SVD-based method is extremely powerful if subsequently a Gaussian smoothing is carried out.

Script 15: SVD-based denoising

```
% Denoising using the SVD-based method
signal = SVD_denoising(signal)

% Denoising using the SVD-based method. Additionally the
% singular values are returned in a vector.
[signal, SVDs] = SVD_denoising(signal)

% Denoising using the SVD-based method. The trunctation
% limit is set to 20 so that only the first 20 singular values
% are kept. Subsequently Gaussian smoothing is applied.
signal = SVD_denoising(signal, 20)
signal = Gaussian_smooth(B, signal, 0.01);
```

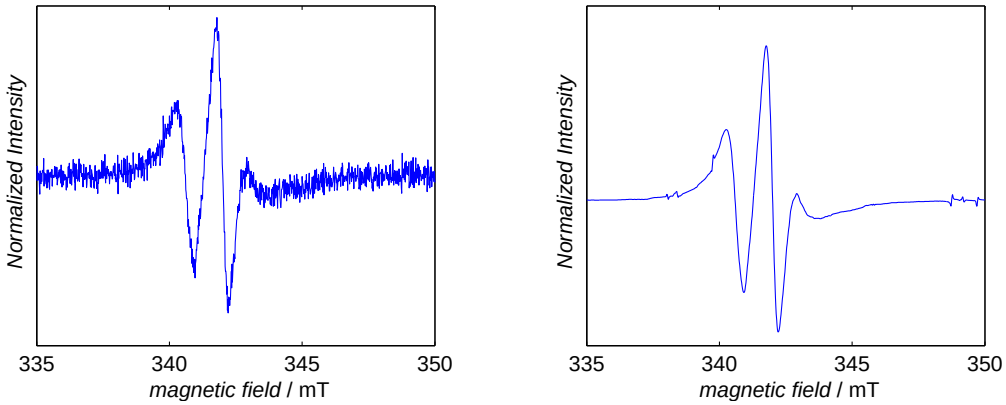


Figure 13: (Left) Noisy signal. (Right) Signal after SVD-based denoising, using the default settings of the `SVD_denoising()` function.

3 Fitting

3.1 General remarks

The fitting module of SpecProFi is the central feature of the program. Global analysis is supported as well as stochastic sampling of initial parameter guesses. In contrast to the data processing function, EasySpin is required for the fitting module [2,3]. The EasySpin syntax is retained. All fitting parameters, usable in SpecProFi, are provided in a `FitOpt` structure. The `Opt` structure (for simulation options) is not mandatory. A basic example for fitting a solid state spectrum is shown in Script 16

Script 16: Basic example for fitting a solid-state spectrum (pepper).

```
% Load the signal
[field, spc] = LoadEPRfile('Sample123');

% EasySpin parameters
Exp.mwFreq = 9.7;
Exp.Range = [min(field), max(field)];
Opt.nKnots = 12;
Sys.S = 0.5;
Sys.g = [2.02, 2.04, 2.12];

% Parameters for fitting
Vary.g = [0.03, 0.03, 0.06];

% SpecProFi options
FitOpt.nLHS = 100;
FitOpt.nIter = 31;

% Call SpecProFi
SpecProFi('pepper', spc, Sys, Vary, Exp, Opt, FitOpt);
```

3.2 Global analysis

For a global analysis multiple data sets are passes to the `specprofi()` main function. If additionally different EasySpin simulation functions ('pepper', 'garlic', 'salt', ...) are required, a cell array of the corresponding strings can be passed.

Script 17: Basic example for fitting a solid-state spectrum (pepper) and a fast-motion spectrum (garlic).

```
% Load the signal
[field, spc] = LoadEPRfile('Sample_liquid');
[field, spc] = LoadEPRfile('Sample_solid');

% EasySpin parameters
Exp.mwFreq = 9.45;
Exp.Range = [min(field), max(field)];
Exp2.mwFreq = 9.7;
Exp2.Range = [min(field2), max(field2)];
Sys.S = 0.5;
Sys.g = [2.02, 2.04, 2.12];
Sys.logtcorr = -11.0;

% Parameters for fitting
Vary.g = [0.03, 0.03, 0.06];

% SpecProFi options
FitOpt.nLHS = 100;
FitOpt.nIter = 31;

% Call SpecProFi
SpecProFi({'garlic','pepper'} , {spc, spc2}, Sys, Vary, {Exp, Exp2}, FitOpt);
```

3.3 All fitting options

3.3.1 Basic fitting options

Option	=	Function
`FitOpt.LHS`	true, false	Enables Latin-hypercube sampling
`FitOpt.nTrials`	n	Number of sampling points
`FitOpt.algorithm`	'name'	Optimization algorithm
`FitOpt.nIter`	n	Number of iterations in optimization
`FitOpt.TolFun`	n	Function tolerance for optimization
`FitOpt.cuttingup`	n, [n1, n2, ...]	Cutting on the right side
`FitOpt.cuttingdown`	n, [n1, n2, ...]	Cutting on the left side
`FitOpt.weightingup`	n, [n1, n2, ...]	Right end of the weighted region
`FitOpt.weightingdown`	n, [n1, n2, ...]	Left end of the weighted region
`FitOpt.weightingfactor`	n, [n1, n2, ...]	Weighting factor for the defined region
`FitOpt.useParallel`	true, false	Enables parallel computing
`FitOpt.AbsScale`	true, false	If disabled, signal parts can be larger one

A standard fitting example is presented in Script 18, using the options presented above.

Script 18: Basic example for fitting a solid-state spectrum (pepper) using more fitting options of SpecProFi.

```
% Load the signal
[field, spc] = LoadEPRfile('Sample123');

% EasySpin parameters
Exp.mwFreq = 9.7;
Exp.Range = [min(field), max(field)];
Sys.S = 0.5;
Sys.g = [2.02, 2.04, 2.12];

% Parameters for fitting
Vary.g = [0.03, 0.03, 0.06];

% SpecProFi options
FitOpt.LHS = false; % Enables Monte-Carlo sampling
```

```
FitOpt.nTrials = 200;
FitOpt.nIter = 51;
FitOpt.weightingdown = 320;
FitOpt.weightingup = 332;
FitOpt.weightingfactor = 2; % Double weighting between 320 and 332 mT
FitOpt.useParallel = true;
FitOpt.algorithm = 'sqp';
FitOpt.AbsScale = false;

% Call SpecProFi
SpecProFi('pepper', spc, Sys, Vary, Exp, FitOpt);
```

For a global analysis the example presented above could look like shown in Script 19.

Script 19: Basic example for fitting a solid-state spectrum (pepper) using more fitting options of SpecProFi.

```
% Load the signal
[field, spc] = LoadEPRfile('Sample_liquid');
[field2, spc2] = LoadEPRfile('Sample_solid');

% EasySpin parameters
Exp.mwFreq = 9.45;
Exp.Range = [min(field), max(field)];
Exp2.mwFreq = 9.7;
Exp2.Range = [min(field2), max(field2)];
Sys.S = 0.5;
Sys.g = [2.02, 2.04, 2.12];
Sys.logtcorr = -11.0;
% Parameters for fitting
Vary.g = [0.03, 0.03, 0.06];

% SpecProFi options
FitOpt.LHS = false; % Enables Monte-Carlo sampling
FitOpt.nTrials = 200;
FitOpt.nIter = 51;
FitOpt.weightingdown = [320, 312];
FitOpt.weightingup = [332, 335];
FitOpt.weightingfactor = [2, 3.5];
FitOpt.useParallel = true;
FitOpt.algorithm = 'sqp';
FitOpt.AbsScale = false;

% Call SpecProFi
SpecProFi({'garlic','pepper'} {spc, spc2} Sys, Vary, {Exp,Exp2} FitOpt);
```

3.3.2 Advanced fitting options

Option	=	Function
`FitOpt.ARes`	n, [n1, n2, ...]	Constant isotropic hyperfine constraints
`FitOpt.gRes`	n	Constant isotropic g-tensor constraints
`FitOpt.equivHFcoupling`	[n1, n2, ...]	Define groups of equivalent nuclei
`FitOpt.smartGrid`	true, false	Enables the smart grid option
`FitOpt.fastGrid`	n	Steps after pre-optimization
`FitOpt.FGalgorithm`	string	Algorithm for post-optimization
`FitOpt.diff_isotopes`	'Iso1, Iso2, ...'	One hyperfine coupling for different isotopes
`FitOpt.isomixtures`	'Iso1, Iso2, ...'	Isotope-mixture with same hyperfine coupling
`FitOpt.nonuniform_field`	vector	Non-uniform field or frequency
`FitOpt.filename`	string	Final output filename
`FitOpt.verbose`	'iter', 'off'	Console information

`FitOpt.ARes` and `FitOpt.gRes`

`FitOpt.ARes` restricts hyperfine tensors to a isotropic value during the fitting procedure. The isotropic values can be also provided for multiple nuclei. The anisotropic components are fitted while the isotropic restrictions are respected.

`FitOpt.gRes` restricts the g-tensor to a isotropic value, provided by the user. `FitOpt.ARes` and `FitOpt.gRes` are especially useful if the isotropic values of the interaction tensors are known (for example from liquid-state cw-EPR data). An example for using isotropic constraints is shown in Script 20.

Script 20: Basic example for fitting a solid-state spectrum (pepper) under isotropic constraints.

```
% Load the signal
[field, spc] = LoadEPRfile('Sample_solid');

% EasySpin parameters
Exp.mwFreq = 9.45;
Exp.Range = [min(field), max(field)];
Sys.S = 0.5;
Sys.g = [2.02, 2.04, 2.12];
Sys.Nucs = 'Cu';
Sys.A = [10, 20, 100];

% Parameters for fitting
Vary.g = [0.03, 0.03, 0.06];

% SpecProFi options
FitOpt.nTrials = 200;
FitOpt.nIter = 51;
FitOpt.gRes = 2.06; % Restriction to giso = 2.06
FitOpt.ARes = 45;   % Restriction to Aiso = 45 MHz
FitOpt.algorithm = 'LevMarq_SR';

% Call SpecProFi
SpecProFi('pepper', spc, Sys, Vary, Exp, FitOpt);
```

FitOpt.equivHFcoupling

`FitOpt.equivHFcoupling` is an option that defines multiple nuclei to be magnetically equivalent. A vector needs to be provided which defines groups of equivalent nuclei:

Script 21: Basic example for fitting a solid-state spectrum (pepper) using more fitting options of SpecProFi.

```
% Define spin system
Sys.Nucs = 'N,H,H,H'
Sys.A = [10, 110; 10, 20; 10, 20; 10, 20];

Vary.A = [10, 10; 8, 8];
FitOpt.equivHFcoupling = [1, 2, 2, 2] % 1 = N-group, 2 = H-group
```

In Script 21 the three hydrogen nuclei were grouped (group 2) into one "magnetic group" in the `FitOpt.equivHFcoupling` vector and therefore defined as magnetically equivalent. In a fitting procedure the same tensor is used for all three hyperfine couplings with the hydrogen nuclei. For this reason, only two tensors are defined in `Vary.A`, one for the nitrogen nucleus and the second for the three hydrogen nuclei.

Script 22: Basic example for fitting a zero-field populations of a spin-polarized triplet.

```
% Load the signal
[field, spc] = LoadEPRfile('Sample_solid');

% EasySpin parameters
Exp.mwFreq = 9.45;
Exp.Range = [min(field), max(field)];
Sys.S = 0.5;
Sys.g = [2.02, 2.04, 2.12];
Sys.Nucs = 'Cu,Cu';
Sys.A = [10, 20, 100; 10, 20, 100];
Sys.logtcorr = -11.0;

% Parameters for fitting
Vary.g = [0.03, 0.03, 0.06];
Vary.A = [10, 10, 30];

% SpecProFi options
FitOpt.nTrials = 200;
FitOpt.nIter = 51;
FitOpt.equivHFcoupling = [1, 1]; % Same hyperfine tensor for both copper

% Call SpecProFi
SpecProFi('pepper', spc, Sys, Vary, Exp, FitOpt);
```

`FitOpt.diff_isotopes` and `FitOpt.isomixtures`

The optional parameter `FitOpt.diff_isotopes` can be used to define a global fit using different isotopes. For example, one spectrum recorded from a ^{17}O-labeled sample, while the other signal is obtained from a ^{16}O-labeled sample. The assumption is that the hyperfine parameters are the same. The nucleus is not provided as `Sys.Nucs`. Instead a cell array with the isotope names is defined as `FitOpt.diff_isotopes` variable.

Script 23: Basic example for fitting two slow-motion spectrum (chili) by using different nitrogen isotopes.

```
% Load the signal
[field, spc] = LoadEPRfile('Sample_liquid_15N');
[field2, spc2] = LoadEPRfile('Sample_liquid_14N');

% EasySpin parameters
Exp.mwFreq = 9.45;
Exp.Range = [min(field), max(field)];
Exp2 = Exp;
Sys.S = 0.5;
Sys.g = [2.02, 2.04, 2.12];
Sys.A = [10, 20, 100];
Sys.logtcorr = -9.0;

% Parameters for fitting
Vary.g = [0.03, 0.03, 0.06];
Vary.A = [10, 10, 30];

% SpecProFi options
FitOpt.nTrials = 200;
FitOpt.nIter = 51;
FitOpt.weightingfactor= [1, 2];
FitOpt.diff_isotopes = {'15N','14N'}; % Global fit with different isotopes

% Call SpecProFi
SpecProFi('chili', {spc, spc2}, Sys, Vary, {Exp, Exp2}, FitOpt);
```

`FitOpt.isomixtures` is an extension to `FitOpt.diff_isotopes`, where a defined mixture of isotopes can be fitted using one set of hyperfine parameters. However, the isotropic contribution of the hyperfine tensor is scaled by the ratio of the gyromagnetic ratio of the isotopes by SpecProFi and needs to be provided by the user. The `FitOpt.isomixtures` is for example useful if two data sets are present. One recorded for example from a sample with pure ^{14}N nuclei and the other from a sample with a 3:4 mixture of ^{14}N and ^{15}N. This example is presented in Script 24.

Script 24: Basic example for fitting two slow-motion spectrum (chili) by using a pure nitrogen isotopes and a isotopes mixture.

```
% Load the signal
[field, spc] = LoadEPRfile('Sample_liquid_14N15N_mix');
[field2, spc2] = LoadEPRfile('Sample_liquid_14N');

% EasySpin parameters
Exp.mwFreq = 9.45;
Exp.Range = [min(field), max(field)];
Exp2 = Exp;
Sys.S = 0.5;
Sys.g = [2.02, 2.04, 2.12];
Sys.A = [10, 20, 100];
Sys.logtcorr = -9.0;

% Parameters for fitting
Vary.g = [0.03, 0.03, 0.06];
Vary.A = [10, 10, 30];

% SpecProFi options
FitOpt.nTrials = 200;
FitOpt.nIter = 51;
FitOpt.weightingfactor= [1, 2];
FitOpt.diff_isotopes = {'15N','14N'}; % Global fit with different isotopes
gyrorato = -1.4 % ratio of the gyromagnetic ratios of 15N and 14N
FitOpt.isomixtures = {1, 4, 3, gyrorato, '15N','14N'}; % 4:3 ratio of 1

% Call SpecProFi
SpecProFi('chili', {spc, spc2}, Sys, Vary, {Exp, Exp2}, FitOpt);
```

FitOpt.smartGrid, FitOpt.fastGrid, FitOpt.FGalgorithm and FitOpt.useParallel

FitOpt.smartGrid, FitOpt.fastGrid and FitOpt.useParallel are tools to make the time consuming semi-stochastic analysis faster. FitOpt.useParallel enables parallel computing. The parallel computing toolbox is only invoked if available and th local parallel pool is used to distribute different sampling points to different CPUs. The FitOpt.smartGrid option can be set to true to enable a "smarter" grid through the parameter hypercube. If the FitOpt.smartGrid is set to true, SpecProFievaluates after half of the optimization steps if it is promising to carry on with the optimization. The algorithm which determines if an optimization process is carried on is rather tedious and can be found in reference [1]. An example for using the FitOpt.smartGrid as well as parallel computing is shown in Script 19.

Script 25: Basic example for fitting a solid-state spectrum (pepper) using the FitOpt.smartGrid option and the FitOpt.useParallel option of SpecProFi.

```
% Load the signals
[field, spc] = LoadEPRfile('Sample_liquid');
[field2, spc2] = LoadEPRfile('Sample_solid');

% EasySpin parameters
Exp.mwFreq = 9.45;
Exp.Range = [min(field), max(field)];
Exp2.mwFreq = 9.7;
Exp2.Range = [min(field2), max(field2)];
Sys.S = 0.5;
Sys.g = [2.02, 2.04, 2.12];
Sys.logtcorr = -11.0;
% Parameters for fitting
Vary.g = [0.03, 0.03, 0.06];

% SpecProFi options
FitOpt.nTrials = 500;
FitOpt.nIter = 120;
FitOpt.useParallel = true;
FitOpt.smartGrid = true;
FitOpt.algorithm = 'sqp';

% Call SpecProFi
SpecProFi({'garlic','pepper'} {spc, spc2} Sys, Vary, {Exp,Exp2} FitOpt);
```

The `FitOpt.fastGrid` is a further option to improve the performance of the analysis, either in speed or chances to find an optimal global solution. The word "fast" in `FitOpt.fastGrid` is rather misleading, as the enabling the `FitOpt.fastGrid` option makes the program not necessarily faster. The `FitOpt.fastGrid` option is tool for a hybrid algorithm combined with a `FitOpt.smartGrid` option. Identical to the `FitOpt.smartGrid` option, the algorithm decides if an optimization process is carried after half of the optimization steps. In contrast to the `FitOpt.smartGrid` option, `FitOpt.fastGrid` uses a different optimization algorithm for carrying on the optimization. This algorithm can be defined by setting `FitOpt.FGalgorithm`. If no algorithm is provided, the Nelder-Mead Simplex algorithm is used for the second optimization. Furthermore, the user needs to define how detailed the second optimization is supposed to be by setting `FitOpt.fastGrid` to a number, which means the number of iterations. An example for using the `FitOpt.fastGrid` is shown in Script 26.

Script 26: Basic example for fitting a solid-state spectrum (pepper) using the `FitOpt.fastGrid` option of SpecProFi.

```
% Load the signals
[field, spc] = LoadEPRfile('Sample_liquid');
[field2, spc2] = LoadEPRfile('Sample_solid');

% EasySpin parameters
Exp.mwFreq = 9.45;
Exp.Range = [min(field), max(field)];
Exp2.mwFreq = 9.7;
Exp2.Range = [min(field2), max(field2)];
Sys.S = 0.5;
Sys.g = [2.02, 2.04, 2.12];
Sys.logtcorr = -11.0;
% Parameters for fitting
Vary.g = [0.03, 0.03, 0.06];

% SpecProFi options
FitOpt.nTrials = 500;
FitOpt.nIter = 120;
FitOpt.useParallel = true;
FitOpt.FGalgorithm = 'interior-point'; % Interior-point for 2nd optimization
FitOpt.fastGrid = 45;       % Use 45 iterations for the 2nd optimization
FitOpt.algorithm = 'sqp';

% Call SpecProFi
SpecProFi({'garlic','pepper'} {spc, spc2} Sys, Vary, {Exp,Exp2} FitOpt);
```

3.3.3 Variable (individual) fitting parameters in global analysis

Some spin system parameters might differ, depending on the experiment. For this reason, some system parameters (`Sys.lw`, `Sys.lwpp`, `Sys.logtcorr`, `Exp.Temperature`, `Exp.mwFreq`, `Exp.ModAmp`, `Exp.mwPhase`) can be fitted individually. Line-widths might be similar for different experiments but are not necessarily the same. Therefore it can be fitted either global or individually. The rotational correlation time changes with temperature and can therefore be fitted individually.

Script 27: Basic example for fitting a liquid-state spectrum (garlic) using individual rotational correlation times.

```
% Load the signals
[field, spc] = LoadEPRfile('Sample_liquid_280K');
[field2, spc2] = LoadEPRfile('Sample_liquid_320K'');

% EasySpin parameters
Exp.mwFreq = 9.45;
Exp.Range = [min(field), max(field)];
Exp2.mwFreq = 9.38;
Exp2.Range = [min(field2), max(field2)];
Sys.S = 0.5;
Sys.g = [2.02, 2.03];
Sys.Nucs = 'N';
Sys.A = [10, 20, 100];
Sys.logtcorr = [-10.5; -11.0];

% Parameters for fitting
Vary.g = [0.01, 0.01];
Vary.A = [10, 10, 20];
Vary.logtcorr = [0.8; 0.8];

% SpecProFi options
FitOpt.nTrials = 200;
FitOpt.nIter = 51;

% Call SpecProFi
SpecProFi('garlic', {spc,spc2}, Sys, Vary, {Exp,Exp2}, FitOpt);
```

While Script 27 shows an example of a global analysis with individually variable rotational correlation time (`logtcorr`), Script 28 shows the example of a global analysis by fitting the line-widths individually.

Script 28: Basic example for fitting a solid-state spectrum (pepper) by using individual line-widths.

```
% Load the signals
[field, spc] = LoadEPRfile('Sample_solid_Xband');
[field2, spc2] = LoadEPRfile('Sample_solid_Qband');

% EasySpin parameters
Exp.mwFreq = 9.45;
Exp.Range = [min(field), max(field)];
Exp2.mwFreq = 33.38;
Exp2.Range = [min(field2), max(field2)];
Sys.S = 0.5;
Sys.g = [2.02, 2.03];
Sys.Nucs = 'N';
Sys.A = [10, 20, 100];
Sys.lw = [0.2, 1.0; 0.4, 0.8];

% Parameters for fitting
Vary.g = [0.01, 0.01];
Vary.A = [10, 10, 20];
Vary.lw = [0.2, 0.2; 0.2, 0.3]; % individually fitting the line-width
Vary.mwFreq = [0.01, 0.02]; % individually fitting the frequencies

% SpecProFi options
FitOpt.nTrials = 200;
FitOpt.nIter = 51;

% Call SpecProFi
SpecProFi('pepper', {spc,spc2}, Sys, Vary, {Exp,Exp2}, FitOpt);
```

3.3.4 Experimental parameter which can be fitted with SpecProFi

Commonly some unknown parameter of the spin system (`Sys`) are optimized in a fitting procedure. However, there might be some experimental parameters which are not determined, like the spin temperature in spin-polarized systems, or not sufficiently determined, like the microwave frequency. Therefore, some experimental (`Exp`) parameters can be fitted. The parameters available for fitting are `Temperature`, `mwFreq`, `ModAmp`, `mwPhase`. It should be noted that for a global analysis each experimental is fitted individually! An example is shown in Script 29.

Script 29: Basic example for fitting a solid-state spectrum (pepper) by using individual linewidths.

```
% Load the signals
[field, spc] = LoadEPRfile('Sample_solid_Xband');
[field2, spc2] = LoadEPRfile('Sample_solid_Qband');

% EasySpin parameters
Exp.mwFreq = 9.45;
Exp.Range = [min(field), max(field)];
Exp2.mwFreq = 33.38;
Exp2.Range = [min(field2), max(field2)];
Sys.S = 0.5;
Sys.g = [2.02, 2.03];
Sys.lw = [0.2, 1.0; 0.4, 0.8];

% Parameters for fitting
Vary.g = [0.01, 0.01];
Vary.mwFreq = [0.01; 0.02]; % Variation of the microwave frequency

% SpecProFi options
FitOpt.nTrials = 200;
FitOpt.nIter = 51;

% Call SpecProFi
SpecProFi('pepper', {spc,spc2}, Sys, Vary, {Exp,Exp2}, FitOpt);
```

3.3.5 Fitting zero-field populations with SpecProFi

Zero-field populations can be fitted by optimizing the (spin) temperature vector (`Exp.temperature`). A typical example is a spin-polarized triplet signal (Script 30).

Script 30: Basic example for fitting a solid-state spectrum (pepper) by using individual linewidths.

```
% Load the signals
[field, spc] = LoadEPRfile('Sample_spinpolarized_triplet');

% EasySpin parameters
Exp.mwFreq = 9.45;
Exp.Range = [min(field), max(field)];
Exp.Temperature = [0.5, 0.2, 0.3];
Sys.S = 1;
Sys.g = 2.02;
Sys.D = [1000, 100];
Sys.lw = [0.2, 2.0];

% Parameters for fitting
Vary.D = [300, 30];
Vary.Temperature = [0.2, 0.2, 0.2]; % Variation of zero-field populations

% SpecProFi options
FitOpt.nTrials = 200;
FitOpt.nIter = 100;

% Call SpecProFi
SpecProFi('pepper', spc, Sys, Vary,Exp, FitOpt);
```

For a global fit, the zero-populations can be either fitted individually for each spectrum or globally. A single `Vary.Temperature` needs to provided for fitting the zero-field populations globally. Otherwise a vector of vectors needs to be provided.

```
% Fitting zero-field populations for two data sets

% Global fitting of zero-field populations
Vary.Temperature = [0.2, 0.2, 0.2];

% individual fitting of zero-field populations
Vary.Temperature = [0.2, 0.2, 0.2; 0.3, 0.2, 0.1];
```

3.3.6 Full list of parameters which can be optimized

List of parameters which can be provided in the `Vary` structure:

- g, gFrame, gStrain
- A, AFrame, AStrain
- D, DFrame, DStrain
- Q, QFrame
- lw, lwpp, lwEndor, HStrain
- J, ee, eeD
- logtcorr
- weight
- nn

Additional experimental parameter which can be provided in the `Vary` structure:

- Temperature
- mwFreq
- ModAmp
- mwPhase

3.3.7 Full list of fitting algorithms

string	Algorithm	Toolbox	Boundaries
'trust-region-reflective'	Trust-region reflective	Yes	Yes
'levenberg-marquardt'	Levenberg-Marquardt	Yes	No
'LevMarq_SR'	Levenberg-Marquardt	No	Yes
'interior-point'	Interior-point	Yes	Yes
'sqp'	Sequential quadratic programming	Yes	Yes
'interior-point'	Interior-point	Yes	Yes
'simplex'	Nelder-Mead simplex	No	No

4 Examples

4.1 Fitting multiple Harmonics

Script 31: Fitting a liquid-state signal by using a global analysis of the field-modulated signal and the integrated signal.

```
% Load the signals and integrate it (spc2)
[B, spc1] = LoadEPRfile('Sample_liquid');
spc1 = baseline_correction(spc1);
spc2  = int_and_diff(spc1, 1);  % Integration of the signal

Exp1= struct('Range',[min(B), max(B)],'mwFreq',9.7646,'Harmonic',1);
Exp2 = Exp1;
Exp2.Harmonic = 0;

% EasySpin parameters
Sys.S = 0.500000  ;
Sys.g = 2.004935  ;
Sys.lwpp = 0.095  ;
Sys.Nucs = '1H,1H';
Sys.A = [4.658200;  14.548000]  ;
Sys.n = [12, 6]  ;

% Parameters for fitting
Vary.lwpp = 0.05 ;
Vary.A = [0.8; 0.8]

% SpecProFi options
FitOpt.nTrials = 50;
FitOpt.nIter = 90;
FitOpt.fastGrid = 150;
FitOpt.weightingfactor = [0.5 1];

% Call SpecProFi
SpecProFi('garlic', {spc1,spc2}, Sys,Vary, {Exp1,Exp1}, FitOpt);
```

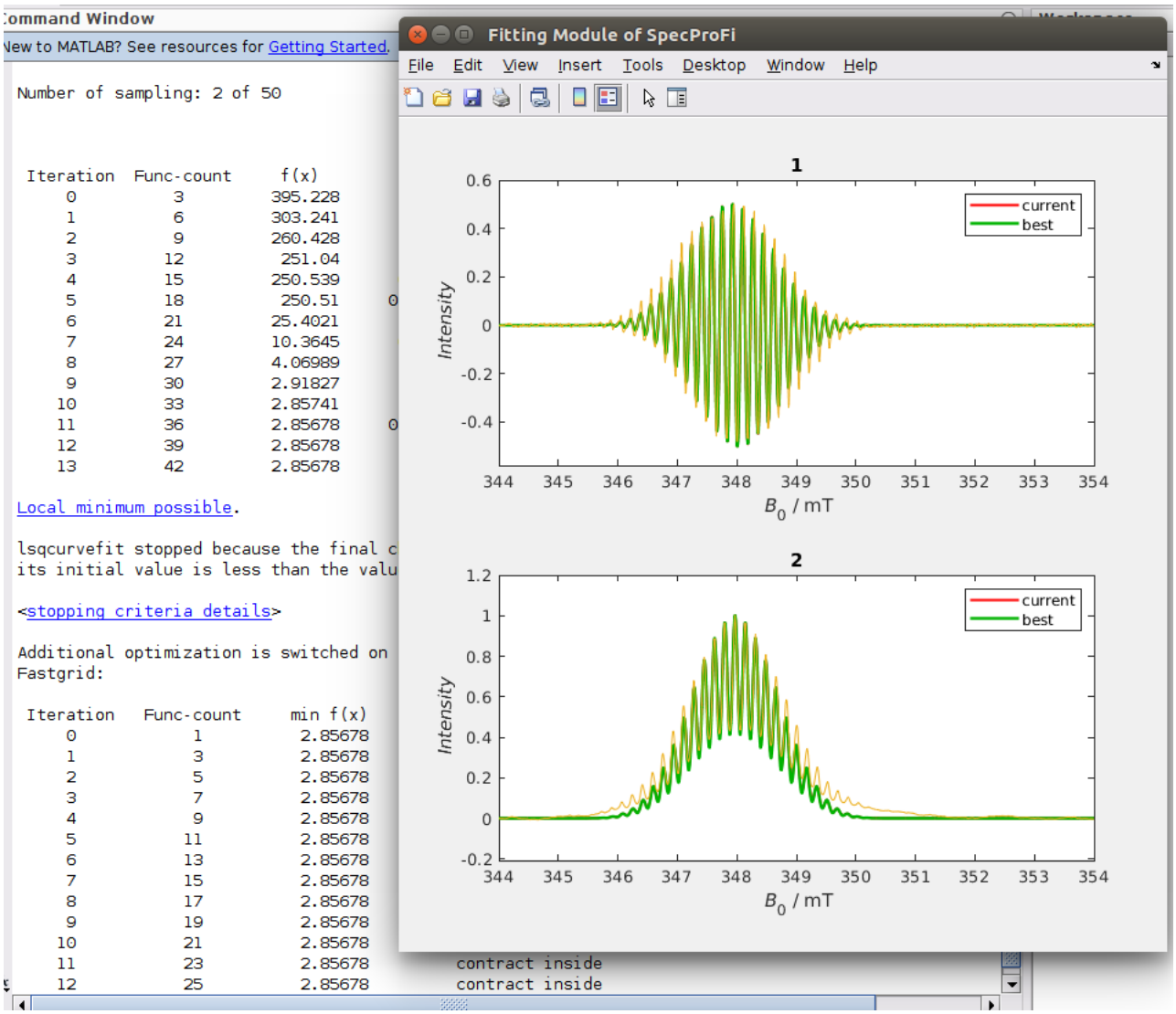


Figure 14: Screenshot of SpecProFi, running with the input shown in Script 31.

4.2 Global analysis of orientation-selective ENDOR data

Script 32: Example for a global analysis of three orientation-selective W-band ENDOR data.

```
% Load the signals
[frq1, spc1] = LoadEPRfile('Sample_ENDOR_X');
[frq2, spc2] = LoadEPRfile('Sample_ENDOR_Y');
[frq3, spc3] = LoadEPRfile('Sample_ENDOR_Z');

% EasySpin parameters
Exp1 = struct('Range',[fmin1 fmax1],'Field',3347.4,'Harmonic',0,...
              length(frq1) 'mwFreq', 93.902579,'ExciteWidth',20);
Exp2 = struct('Range',[fmin2 fmax2],'Field',3348.6,'Harmonic',0,'nPoints',...
              length(frq2),'mwFreq', 93.90276,'ExciteWidth',10);
Exp3 = struct('Range',[fmin3 fmax3],'Field',3350.7,'Harmonic',0,'nPoints',...
              length(frq3),'mwFreq', 93.897446,'ExciteWidth',20);

% Radical 1
Sys1 =  struct('S',1/2,...
               'Nucs','13C', 'A',[15.083357 11.607984 13.030755], ...
               'AFrame', [1 1  1], 'weight', 0.610729, ...
               'g',[2.00427 2.0036 2.0022],'lwEndor',0.349645);

% Radical 2
Sys2 = struct('S',1/2,...
              'Nucs','13C', 'A',[0.854952 1.725230  23.429808], ...
              'AFrame', [0.019530  0.244436  0.746466],...
              'g',[2.00427 2.0036 2.0022],'lwEndor',0.28, 'weight',0.3463);

SimOpt  = struct('nKnots',40,'Verbosity',0,'Intensity','on', ...
                 'Method','perturb2');

% Parameters for fitting
Vary1.AFrame = [1 1 1];
Vary1.A = [3 3 3];
Vary2.A = [3, 2, 2];
Vary2.weight = 0.2;

% SpecProFi options
FitOpt.nTrials = 200;
FitOpt.nIter = 80;

% Call SpecProFi
SpecProFi('salt',{spc1,spc2,spc}, {Sys1,Sys2}, {Vary1,Vary2}, ...
          {Exp1,Exp2,Exp3}, SimOpt, FitOpt);
```

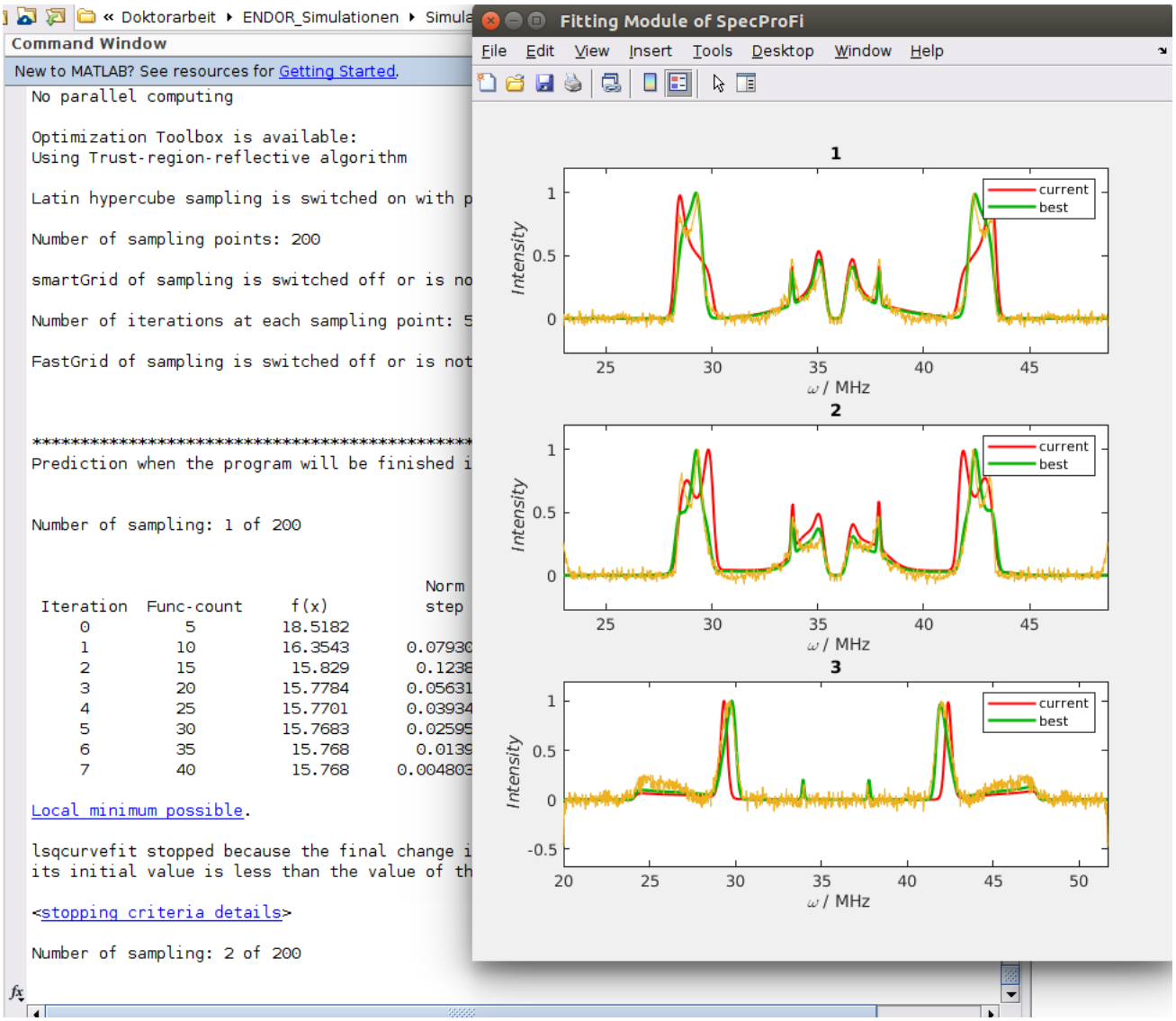


Figure 15: Screenshot of SpecProFi, running with the input shown in Script 32.

References

[1] Stephan Rein. *Development of advanced simulation and analysis programs for EPR spectroscopy.* dissertation, University of Freiburg, 2019.

[2] Stefan Stoll and Arthur Schweiger. EasySpin, a comprehensive software package for spectral simulation and analysis in EPR. *J. Magn. Reson.*, 178:42–55, 2006.

[3] Stefan Stoll and R. D. Britt. General and efficient simulation of pulse EPR spectra. *Phys. Chem. Chem. Phys.*, 11:6614–6625, 2009.