

Spec1d

A versatile set of tools to analyse 1d data sets in MATLAB

A collection of routines designed to facilitate the analysis of one dimensional (i.e. x, y, error) data sets from the command line in MATLAB. Data can be loaded into a structure, known as a **spec1d** object, and then processed with simple one-line commands. Liberal use is made of the operator overload feature of MATLAB5 so that operators such '+', '*' are redefined to work on 1d spectra, including the correct handling of error propagation. Using **spec1d** it is possible to perform complex data analysis, including non-linear least-squares fitting, in a very efficient manner. As it makes use of the load and function libraries from MFIT/MVIEW, it is completely general, and can be used to analyse data from more or less any source.

Requires MATLAB 5.1 or higher.

Introduction

Command Summary

Installation

Case Studies

Version 2.0, February 2001

Des McMorro and Henrik Rønnow

Introduction

Spec1d is presently built on the following class structure:

```

x-data      :      s.x
y-data      :      s.y
y error     :      s.e
x-label     :      s.x_label
y-label     :      s.y_label
datafile    :      s.datafile
y-fit       :      s.yfit

```

Spec1d objects can either be singular, or arrays. In this way it is possible to store and perform data analysis on many files (in fact whole data sets) using one line commands.

Note: The handling of errors in spec1d is designed not to be restricted to the case where the errors are determined by counting statistics. You are advised, however, to check the routines to make sure that they are applicable to the problem you are trying to treat.

The easiest way to become familiar with **spec1d** is to study a number of examples.

Reading data files: use of the **loads** command

Example 1: Reading data stored in x, y, error format, and fitting it to a Gaussian peak

Example 2: Reading data from a formatted data file (ILL, SPEC, RISØ, etc)

Example 3: Reading multiple files, or multiple scans from the same file

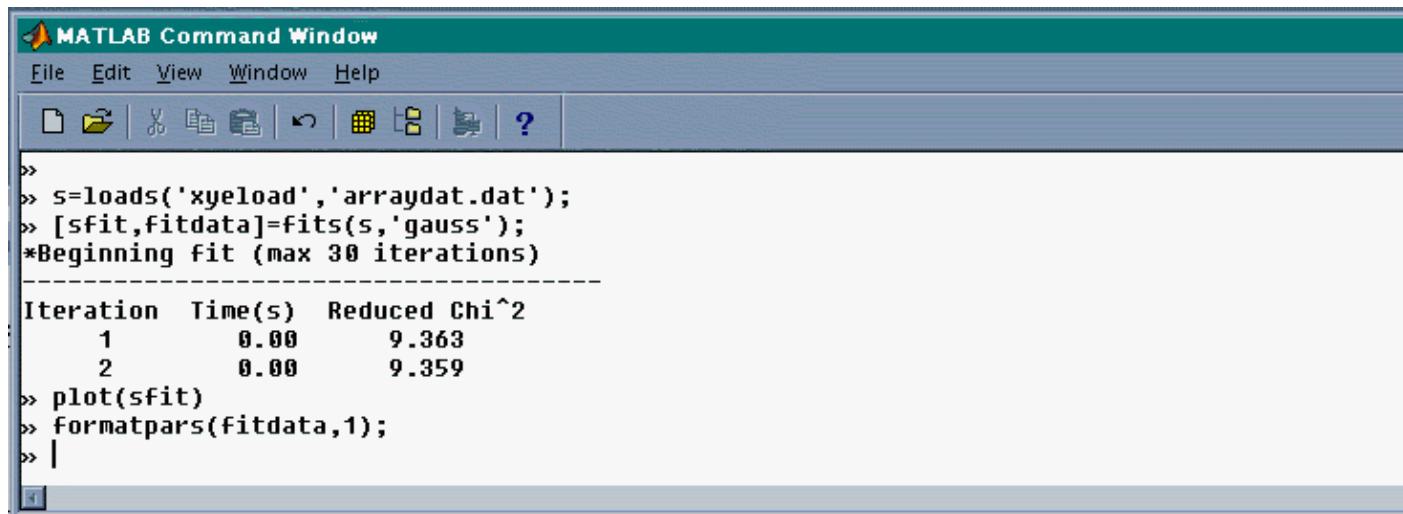
Example 1: Reading data stored in x, y, error format, and fitting it to a Gaussian peak

Spec1d has built-in filters to load most types of data files. If the filter does not exist, then it is either easy to

create a new filter, or to convert the data to x, y, error format, and then read in using the 'xyload' filter.

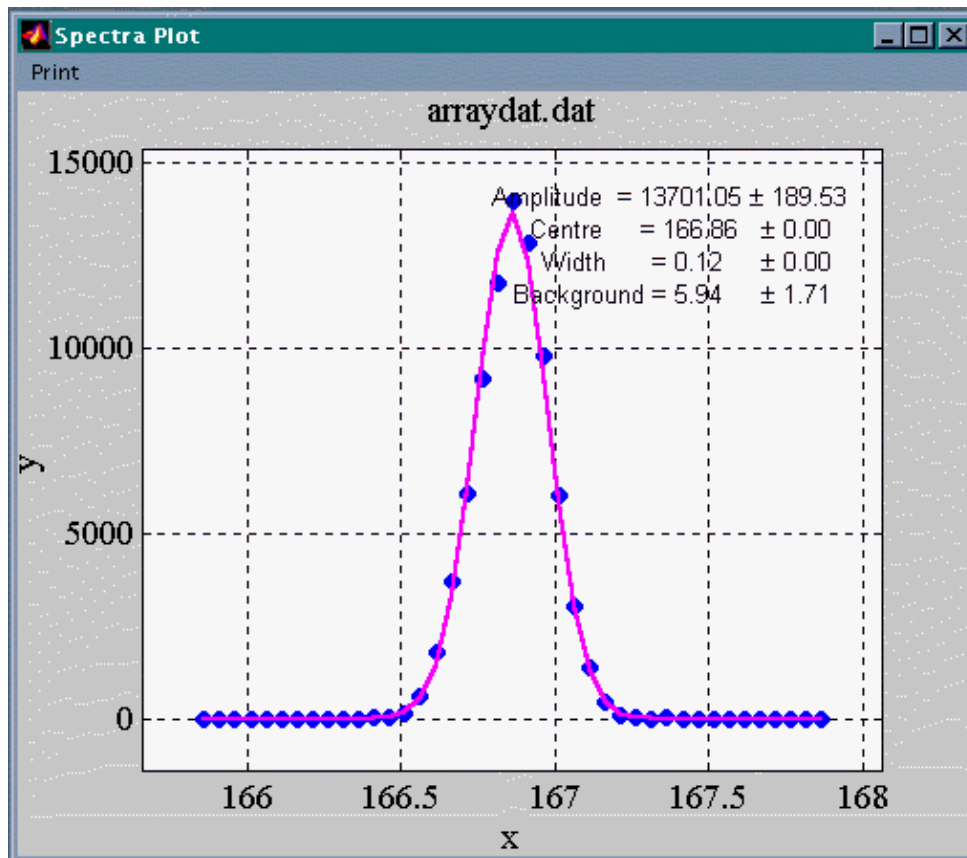
For example, the following commands can be used to load the file arraydat.dat (x, y and error stored in

column form), fit it to a Gaussian, and to plot it:



```
MATLAB Command Window
File Edit View Window Help
[Icons]
>>
>> s=loads('xyload','arraydat.dat');
>> [sfit,fitdata]=fits(s,'gauss');
*Beginning fit (max 30 iterations)
-----
Iteration   Time(s)   Reduced Chi^2
      1         0.00         9.363
      2         0.00         9.359
>> plot(sfit)
>> formatpars(fitdata,1);
>> |
```

which produces the figure



The code generated the plot in the following way:

- The **loads** function is used to load the data (x, y and error) from the file 'arraydat.dat' into the **spec1d** object `s`.
- The spectrum in `s` is then fitted to a Gaussian lineshape using the **fits** function. As no initial parameter values were given, the fitting routine estimates them from a moment analysis using the function **peakm**. The parameters of the fit are returned in the structure `fitdata`, and the fitted values and original data in `sfit`.
- **plot** is used to plot the data and fit.
- Finally, the fit parameters in `fitdata` are added to the plot using **formatpars**.

In this example, each command was typed at the matlab prompt. They can of course be placed in a file, which is then run from the command line. This allows the steps used in a complex data analysis to be stored for editing, and also as a record of what was done.

Both **loads** and **fits** are much more flexible than this example suggests, as will be made clear in the other examples. Check the help supplied with each function for details.

Example 2: Reading data from a formatted data file (ILL, SPEC, RISØ, etc)

Filters for most commonly encountered data files are available in the MFIT/MVIEW load library. Using these filters it is possible to specify the x and y variables, name of the column containing the errors (if it exists), the monitor, and even a normalisation value. The **loads** routine stores the values of these variables, so that in subsequent calls to **loads** only the variables that change (for example the data file name or scan number) need to be specified.

Let us imagine that we want to load two TASCUM files from Risø into separate spectra, and then overplot them. The desired x and y variables are OM and I, the errors are calculated from the counting statistics, and the monitor is MON. The name of the data filter in this case is 'tasbatch', and the data file is 'beta0007.dat'.

Example 3: Reading multiple files, or multiple scans from the same file

Command summary

Command	Description	Examples
+	Addition of the y values of spectra, or addition of constants to the x and y values.	<p>Add together the y values in s1, s2, and s3: <code>>>stot=s1+s2+s3;</code></p> <p>Add 10 to the y values in s8: <code>>>s9=s8+10;</code></p> <p>Add 10 to the y values, and 2 to the x values: <code>>>s9=s8+[2 10];</code></p> <p>Note: The spectra to be added must have identical x axes; otherwise use combine.</p>
-	Subtraction of the y values of two spectra, or subtraction of constants from the x and y values.	<p>Subtract the y values in s2 from those in: <code>>>s3=s2-s1;</code></p> <p>Subtract 10 from the y value in s1: <code>>>s2=s1-10;</code></p> <p>Subtract 20 from the x value in s10: <code>>>s11=s10-[20 0];</code></p> <p>Note: Can only subtract spectra if they have identical x axes.</p>
*	Multiplication of the y values of two spectra, or multiply the x or y values by a constant	<p>Multiply the y values of s2 and s1: <code>>>s3=s2*s1;</code></p> <p>Multiply the y value in s1 by 10: <code>>>s2=s1*10;</code></p> <p>Multiply the x value in s5 by 2 and the y value by 3.5: <code>>>s7=s5*[2 3.5];</code></p> <p>Note: When multiplying spectra, they must have identical x axes.</p>
/	Division of the spectra of the y values of two spectra, or divide the x or y values by a constant	<p>Divide the y values of s2 by those in s1: <code>>>s3=s2/s1;</code></p> <p>Divide the y values in s1 by 10: <code>>>s2=s1/10;</code></p> <p>Divide the x values in s5 by 2 and the y value by 3.5: <code>>>s7=s5/[2 3.5];</code></p> <p>Note: When dividing two spectra, they must have identical x axes.</p>
combine	Combine averages the y values in two or more spectra. The x tolerance and combination method are optional. The default method assumes that errors are determined by counting statistics.	<p>Average s1, s2 and s3 if the x points are closer than 0.01: <code>>>s4=combine(0.01,s1,s2,s3);</code></p> <p>Average all of the spectra in the array of spec1d objects stotal. No tolerance is specified; the default value is eps: <code>>>sc=combine(stotal);</code></p>
cut	Remove data over specified range(s) in x	<p>Cut s1 outside of the range 101 to 102 in x <code>>>r=cut(s1,[101 102]);</code></p> <p>Cut s1 in the x range 101–102 and 109–110</p>

spec1d

		<code>>>r=cut(s1,[102 101],[110 109]);</code>
display	Display the x, y and error values in a spectra	Print the x, y and error values of s2 in the Matlab command window <code>>>disp(s2);</code>
dxdy	Differentiate a spectra	Differentiate spectra s4: <code>>>sd=dxdy(s4);</code>
extract	Extract the x, y, and error values to arrays	Extract the data from s10 to arrays x, y, error: <code>>>[x,y,e]=extract(s10);</code>
fits	Perform a non-linear least-squares fit on of a specified model to the data in a spectra	Fit a Gaussian to s1, with automatic estimation of start values of parameters. All parameters are allowed to vary, and the default values for the fit control parameters are used: <code>>>[sfit,fitpars]=fits(s1,'gauss');</code> Same as above, but with initial parameters specified and background (parameter 4) fixed: <code>>>[sfit,fitpars]=fits(s1,'gauss',[0.9 0.1 0.2 0.1],[1 1 1 0]);</code> As above, but with the fit control parameters specified: <code>>>[sfit,fitpars]=fits(s1,'gauss',[0.9 0.1 0.2 0.1],[1 1 1 0],[0.001 20 0.01]);</code>
getfield	Get the values of specified fields	Obtain the x values from d24: <code>>>x=getfield(d24,'x');</code> Obtain the x and y values from d24: <code>>>[x,y]=getfield(d24,'x');</code> Obtain the x values and the x_label from d24: <code>>>[x,xlab]=getfield(d24,'x','x_label');</code> Note: the number of output variables must equal the number of fields requested.
horzcat	Create an array of spectra	Form an array of the spec1d the objects s4, s5, s6: <code>>>sout=[s4 s5 s6];</code>
interpolate	Interpolate a spectrum to new x vales	Interpolate s10 to the x values specified in the array xnew: <code>>>snew=interpolate(s10,xnew);</code>
loads	Load data from a file into a spectra	Load scan number 14 from the SPEC file ce2.dat: <code>>>s1=loads('specbatch','ce2.dat',X=Theta,Y=Exp_Hutch,M=Mon,S=14');</code> Load and combine scans 14 to 20: <code>>>s1=loads('specbatch','ce2.dat',X=Theta,Y=Exp_Hutch,M=Mon,S=[14++20]);</code> Load scans 14 to 20 into an array: <code>>>s1=loads('specbatch','ce2.dat',X=Theta,Y=Exp_Hutch,M=Mon,S=[14:20]);</code> Load data from the ILL file ho001.dat: <code>>>s1=loads('illbatch','ho001.dat',X=OM,Y=I1,M=Mon);</code> Load and combine ho001.dat to ho009.dat: <code>>>s1=loads('illbatch','ho00[1++9].dat',X=OM,Y=I1,M=Mon);</code> Load ho001.dat to ho009.dat into an array:

spec1d

		<pre>>>s1=loads('illbatch','ho00[1:9].dat', X=OM,Y=I1,M=Mon');</pre>
mapplot	Make a 2D plot of the data in an array of spectra. Note: this is a visualisation tool, and not for data analysis.	Plot array stot (21 spectra): <pre>>>mapplot(stot);</pre> Plot and smooth using a Gaussian convolution of width 0.1; <pre>>>mapplot(stot,[1:21],0.1);</pre>
normalise	Normalise the y data in a spectra to a specified value	Normalise y data in s2 to 10.5 <pre>>>s5=normalise(s2,10.5);</pre>
peakm	Calculate peak properties using the method of moments	Perform a peak analysis on s2: <pre>>>stats=peakm(s2);</pre> Perform peak analysis on array sall: <pre>>>stats=peakm(sall);</pre>
peakt	Calculate the integrated intensity (and its error) using trapezoidal integration	Trapezoidal integration of peak in s3: <pre>>>stats=peakt(s3);</pre> Trapezoidal integration of array stot: <pre>>>stats=peakt(stot);</pre>
plot	Basic plotting routine for displaying the contents of one or several spectra	Plot s3: <pre>>>plot(s3);</pre> Overlay s1, s2 and s3: <pre>>>plot(s1 s2 s3);</pre> Overlay s1, s2 and s3 on a loglog plot: <pre>>>plot(s1 s2 s3,'loglog');</pre> Overlay s1, s2 and s3 on a semilogy plot: <pre>>>plot(s1 s2 s3,'loglog');</pre> Overlay spectra in array st: <pre>>>plot(st);</pre>
rebin	Rebin the data in a spectra to a new x axis	
setfield	Set the values of specific fields	Set the x values in d24 to array xn: <pre>>>d24=setfield(d24,'x',xn);</pre> Set the datafile variable (i.e. title of graph) <pre>>>d24=setfield(d24,'datafile','Great data');</pre>
smooth	Smooth the data using a 1D Gaussian convolution	Convolute s with a Gaussian of variance 0.2: <pre>>>s=smooth(s,0.2)</pre>
spec1d	Create a new spec1d object	For a structure r <pre>>>r=spec1d(r)</pre> converts r to a spec1d object
vertcat	Append the data from several spectra into one	Create a single spec1d object from s1, s2 and s3: <pre>>>snew=[s1; s2; s3];</pre>

\section{Reference}

\subsection{Loading spectra}

Spectra can be loaded using the command file loads. This makes use of the standard MFIT/MVIEW load routines to load the data into a data structure.

For example, the SPEC data file 'spec.01' maybe loaded into the data structure 's1' using the following command

```
>>s1=loads('specbatch','spec.01,X=K,Y=Detector,S=3');
```

Here 'specbatch' specifies the standard MFIT/MVIEW load routine for SPEC type data files, and the line 'spec.01,X=K,Y=Detector,S=3' causes the 3rd scan from the file 'spec.01' to be loaded with the x variable equal to 'K' and the Y variable set to 'Detector'.

s1 is a data structure defined in 'spec1d.m' to be

```
s1.x      : x-data
s1.y      : y-data
s1.e      : error on y
s1.x_label : Label for x-axis
s1.y_label : Label for y-axis
s1.datafile : Data file from which the scan was loaded
s1.yfit    : Fitted value of data
```

\subsection{Basic plotting}

A basic plot of the spectra can be generated with the plot command, eg.

```
>>plot(s1);
```

Two or more spectra can be overlayed

```
>>plot(s1,s2,s3);
```

Most properties of the graph can be edited by clicking on the appropriate feature. Left click on the lables to edit them. Right click on the lines to edit them.

\subsection{Simple operations}

Simple arithmetic operations may be performed on the spectra using the plus '+', minus '-', element-by-element multiplication '.*', and simple element-by-element divide './'.

Note: these operations require that the spectra have identical x-values. If this is not the case then the spectra have to be first interpolated, or in the case of '+' 'combine' may be used.

For example, suppose we wish to load scans 3, 4 and 5 from file 'spec.01' and then process them using simple arithmetic.

First load the scans into workspaces:

```
>>s3=loads('specbatch','spec.01,X=K,Y=Detector,S=3');
>>s4=loads('specbatch','spec.01,X=K,Y=Detector,S=4');
>>s5=loads('specbatch','spec.01,X=K,Y=Detector,S=5');
```

We can now add the three spectra and view the result by typing:

```
>>r=s3+s4+s5;
>>plot(r);
```

Alternatively s5 could be subtracted from the sum of s3 and s4.

```
>>r=s3+s4-s5;
>>plot(r);
```

The operators '.*' and './' work in a similar way, e.g.

```
>>r=s3.*s4;
```

or even

```
>>r=2.*s4-s3;
```

It is possible to append spectra as follows

```
>>r=[s1;s2;s3;s4];
```

\subsection{Other operations}

These will be added to with time.

See the help section of the individual commands.

Presently supported commands include:

\subsubsection{combine}

This is used to combine data points when averaging spectra that do not have the same x-values, e.g.,

spec1d

```
>>r=combine(0.01,s1,s2,s3);
```

Averages points in spectra s1, s2, and s3 that do not differ by 0.01 in their x values. This can also be used as a crude rebinning of one spectra

```
>>r=combine(0.01,s1);
```

```
\subsubsection{cut}
```

```
\subsubsection{disp}
```

```
\subsubsection{extract}
```

```
\subsubsection{fits}
```

```
\subsection{formatpars}
```

```
\subsection{gfields}
```

```
\subsubsection{peakm}
```

```
\subsubsection{peakt}
```

```
\subsection{rebin}
```

```
\subsection{sfields}
```

```
\subsubsection{transform}
```

```
spectrum_out=transform(transform_function,spectrum_in,args)
```

Transform spectrum according to the transform function, e.g.,

```
>>r=transform('normalise',spectra,1)
```

Currently available transform functions include:

- r=transform('normalise',spectra,normalisation_value);
- r=transform('dydx' ,spectra);

Installation

To use **spec1d** you need to:

- Use MATLAB 5.1 or higher
- Download latest version of MFIT/MVIEW load routines from <http://www.ill.fr/tas/matlab/ftp/load.tar.gz>
- Download latest version of MFIT/MVIEW func routines from <http://www.ill.fr/tas/matlab/ftp/funcs.tar.gz>
- Obtain the latest spec1d files from

You should create a directory called '**spec1d**' in a directory which should be entered in your matlab path. Below this directory create another directory called **@spec1d**. Place the files formatpars.m, loads.m, specdfdp.m, and speclsqr.m in the **spec1d** directory, and all other files in **@spec1d**.

Obtaining latest Matlab stuff (MFit/MView/Rescal/SSym/load/funcs/nllsq) from <http://www.ill.fr/tas/matlab/doc/>

Case Studies

The examples here are designed to illustrate the versatility of spec1d in attacking complex data analysis problems in a flexible and transparent way.

Case Study 1: Fitting the order parameter of the magnetic transition in MnPS3

Case Study 1: Fitting the order parameter of the magnetic transition in MnPS3