Documentation for Spectra.h and Spectra.c

Steven Andrews, © 2003
See the document "LibDoc" for general information about this and other libraries.

```
#include "string2.h"
#define MAXARG 20

#include "string2.h"
#define MAXARG 20

typedef struct spect       {
     char name[STRCHAR];
     char file[STRCHAR];
     char desc[STRCHAR];
     char xunit[STRCHAR];
     char yunit[STRCHAR];
     char color[STRCHAR];
     int n;
     int cmplx;
     float *x;
     float *y; } *sptr;

sptr SpectAlloc(char *name,char *file,char *desc,char *xunit,char *yunit);
void SpectFree(sptr s);
int LoadSpect(sptr *spt,char *fname,int xcol,int ycol,int skip);
int SaveSpect(sptr s);
void SpectRange(sptr s,float *xa,float *xb,float *ya,float *yb,int fn);
void TypeSpect(sptr s);
void PlotSpect(sptr s);
int SpectMath(sptr s1,sptr s2,sptr *s3ptr,char *fn,float k);
int SpectMath2(sptr s,sptr *ansptr,float *num,int nn,char *fn,char *fn2);
```

Requires: `<stdio.h>`, `<stdlib.h>`, `<string.h>`, `<math.h>`, `"math2.h"`, `"Rn.h"`, `"Cn.h"`, `"Plot.h"`, `"DiskIO.h"`, `"RnSort.h"`

Example program: `SpectFit.c`

Written 9/16/98; moderate testing. Thoroughly proofread 6/99. Slight modification 8/99. Works with Metrowerks C. Added Hankel transform 1/02. Removed spectrum equation, added complex flag, and added some error reporting 2/7/02. The updating is only partially complete, so the new version is Spectra2.c and the original is unchanged.

This library is designed to be used for spectral fitting and analysis, or for the processing of similar data (such as time resolved measurements). While it was largely developed for use with the SpectFit program, it is a general purpose library for the manipulation of any set of *x*,*y* data. All the routines are platform independent with the exception of `PlotSpect`, which only works on a Macintosh.

The type `struct spect` defines a spectrum. The members `name`, `file`, `desc`, `xunit`, and `yunit`, and `color` are fairly self-explanatory, storing, respectively, the spectrum name, a file name if it has been saved, a description, the plotting color using the `Plot.c` codes, and the units for the *x* and *y* values. The first character of the color is all that is used for a

real spectrum; for complex spectra, the first character is used for the real component and the second character for the imaginary component. n is the number of data points in the spectrum. cmplx is a flag which is 0 if the *y* values are real and 1 if they are complex. x and y are arrays for the data, with n elements each if the complex flag is not set and n elements in the *x* array and 2n elements in the *y* array if the flag is set. In the latter case, the even *y* elements are the real components and the odd elements are the corresponding imaginary values. While there are strings allocated for *x* and *y* units, they are not checked or updated in any of the routines here.

In general, spectra are assumed to have all structure members allocated and in good order. Also, it is assumed that n is at least 1 and that x and y are allocated, initiallized, and sorted with increasing x values. Routines here that do not assume that x and y are set up are SpectAlloc (which returns a spectrum with NULL x and y), SpectFree, TypeSpect, and PlotSpect. If spectra are properly initiallized, every function should return either a correct answer or an error code.

SpectAlloc allocates space for a spectrum and initiallizes members as possible, returning a pointer to the spectrum, or NULL if allocation was unsuccessful. If the input strings are defined, they are copied into the new spectrum (the input strings may be freed afterwards if necessary without affecting the spectrum); alternatively, some or all NULL values may be entered, in which case the spectrum is initiallized with empty strings. The x and y members of the spectrum are returned as NULL, so they need to be allocated and assigned elsewhere.

SpectFree frees a spectrum and any data that it might include. If the arrays were already freed, the pointers should be set to NULL.

LoadSpect loads a real valued spectrum from disk, stored in a table with skip lines of header, where the *x* data are read from the xcol column (numbered starting with 1) and the *y* data are read from the ycol column. It uses the LoadData3 routine from DiskIO.c, and so has the same file format requirements. spt points to a spectrum and should be sent in either pointing to NULL or to a spectrum; in the latter case, the spectrum will be freed automatically. If the loading worked, the spectrum is returned in *spt; otherwise *spt is set to NULL and an error code is returned, where the error code is one of the values listed in the DiskIO.c documentation (possible values are 0, 1, 2, 4, 5, and 6). If loading works, spectra are returned sorted with increasing *x* values.

SaveSpect saves a real or complex spectrum to disk, giving a file with no header and either a pair of columns for *x* and *y* data if the spectrum is real, or three columns for *x*, Re(*y*), and Im(*y*) if the spectrum is complex. If the spectrum has a file name, it is used; otherwise the user is asked for a file name. The routine returns one of the error codes from DiskIO.c.

SpectRange returns the range of the spectrum if fn=0, or the inside of ranges of the spectrum and xa,xb,ya,yb if fn=-1, or the outside of the ranges if fn=1. If s is complex, then the spectral *y* range includes both real and imaginary components.

TypeSpect displays the header of a spectrum on the standard output, as well as the first and last data points. A NULL spectrum or a spectrum without *x* and *y* data is allowed.

PlotSpect plots a spectrum to the graphics window, using Plot.c commands. The color of the spectrum is the value of the first character of the color string; if s is complex, the second character of the string is used for the imaginary component. A NULL spectrum or a spectrum missing *x* or *y* data results in nothing plotted.

SpectMath does simple math with spectra. Inputs are the spectra s1 and s2 and the number k, while the ouput is pointed to by s3ptr. s1 is always required but s2 and k are only required for certain functions. If s3ptr is sent in pointing to NULL (s3ptr should not be NULL) a new spectrum is created, otherwise the existing one is overwritten (be careful not to send in an uninitialized pointer). fn is a string which identifies the math function to be done. If the procedure is successful, it returns 0

and the output spectrum is set with a blank name and blank file name, and with the description, units, and color copied from s1. When possible, output data points are at the s1 point locations. The description is appended with *fn, although units are not updated. If the procedure fails, an error code is returned, listed below; also, if s3ptr pointed to NULL, it is returned that way and nothing needs freeing, otherwise the previous spectrum pointed to by s3ptr was probably changed. It is always permissible for s1 and s2 to point to a single spectrum, but *s3ptr needs to be distinct. Sizes of data sets are modified as needed and spectra are linearly interpolated or extrapolated as needed. Math cannot be done with two spectra in which only one is complex. Virtually all functions return spectra of the same type that are entered, the exceptions being complex, real, and imag. Undefined numbers, such as division by zero or the square root of a negative number using real spectra, are set to an answer of zero.

*fn	s2 or k	type	returns
copy		r,c	copy of s1
smooth		r	smooths s1 with k points on each side
log		r,c	base 10 log of s1
exp		r,c	exp(s1)
ln		r,c	natural log of s1
10^s		r,c	10^s1
sqrt		r,c	square root of s1
s*x or x*s		r,c	multiplies s1 by its x values
s/x		r,c	divides s1 by its x values
deriv1, deriv		r	first derivative of s1
integ	k	r	integral of s1, set to 0 at *x*=k
deriv2		r	second derivative of s1
xderiv1		r	x weighted first derivative of s1
xderiv2		r	x weighted second derivative of s1
s+k or k+s	k	r,c	s1+k, where k is real
s-k	k	r,c	s1-k, where k is real
k-s	k	r,c	k-s1, where k is real
s*k or k*s	k	r,c	s1*k, where k is real
k/s	k	r,c	k/s1, where k is real
s/k	k	r,c	s1/k, where k is real
s^k	k	r,c	s1^k, where k is real
s*s	s2	r,c	s1*s2
s/s	s2	r,c	s1/s2
s+s	s2	r,c	s1+s2
s-s	s2	r,c	s1-s2
merge	s2	r,c	combines s1 and s2, with smooth transition
log x		r,c	x of s3 is base 10 log of x of s1
ln x		r,c	x of s3 is natural log of x of s1
exp x		r,c	x of s3 is exp(x of s1)
10^x		r,c	x of s3 is 10^(x of s1)
sqrt x		r,c	x of s3 is square root of x of s1
x+k or k+x	k	r,c	x of s3 is (x of s1)+k
x-k	k	r,c	x of s3 is (x of s1)-k
k-x	k	r,c	x of s3 is k-(x of s1)
x*k or k*x	k	r,c	x of s3 is k*(x of s1)
x/k	k	r,c	x of s3 is (x of s1)/k
k/x	k	r,c	x of s3 is k/(x of s1)
x^k	k	r,c	x of s3 is (x of s1)^k
complex	(s2)	r	converts s1 and s2, if given, to complex

real		c	real part of s1
imag		c	imaginary part of s1
fourier	k	c	fourier transform of s1, starting at k
invfourier	k	c	inverse fourier transform of s1, starting at k
ftpower		r	fourier power spectrum of s1, starting at k
fft	k	c	fast fourier transform of s1, starting at k
invfft	k	c	inverse fft of s1, starting at k
hankel	k	r	hankel transform of s1, oversampled by k
noise	k	r,c	Gaussian noise with std. dev. k
convolve	s2	r	Convolves s1 with kernal s2

Possible error codes

1 out of memory
6 missing inputs to function
7 command not recognized
8 complex spectra not supported for this function
9 divide by zero
10 only one spectrum is complex
11 incompatible spectra x values
12 real spectra not supported for this function
15 data are not evenly spaced
44 argument out of bounds
999 an impossible situation occurred

For writing more functions, it is helpful to know how the variables are set up initially. At the start, s3 is set up to be similar to s1, with the same *x* values, the same number of points, the same type (complex or not), the same color, with the *y* array allocated to the correct size, and with the description updated. Before running the individual functions, several variables are defined for convenience:

spectrum 1:	s1	n	x1	y1	cmplx
spectrum 2:	s2	n2	x2	y2	
spectrum 3:	s3	n	x3	y3	cmplx

The only one of these variables that is used after running the individual functions is s3; however, it is still a good idea to preserve these variables in the functions, and update them if necessary. Also, er and sort are initially 0 and are used, respectively, to define any error code to indicate that the *x* vector of s3 needs to be re-ordered. There are also several generic variables for use as needed, which are: i, j, i2, f1, f2, f3, v1, v2, sz1, and sz2; none of these are allocated at the beginning or freed at the end. num may be used to pass numbers to SpectMath2.

SpectMath2 is similar to SpectMath, but does different math and has different input variables. s is the input spectrum and the answer is returned in a spectrum pointed to by *ansptr. If ansptr points to a spectrum initially, that spectrum is automatically freed; alternatively, it may point to a NULL spectrum initially, although ansptr should not be NULL. fn is the name of the function, fn2 is a string parameter that modifies the function, num is an array of numbers from 0 to nn, and s is the input spectrum. More precisely, nn is the number of input numbers; for the "baseline" function, nn should be set to 0, but num should be allocated for at least 2 values for the output numbers. Using the notation f_j for num[j], the functions are:

*fn	*fn2	type	description
new	1	r	spectrum of all 1's, from f_0 to f_1, with step size f_2

	x		spectrum with y=x, from f_0 to f_1, with step size f_2
copy		r,c	copies s, from f_0 to f_1, optional step size f_2
zerofill	left	r,c	expand s by factor f_0, filling left with zeros
	right		expand s by factor f_0, filling right with zeros
	ends		expand s by factor f_0, filling ends with zeros
baseline	left	r	baseline correct s to left edge, return f_0 as offset
	right		baseline correct s to right edge, return f_0 as offset
	ends		correct s to ends, return f_0 as slope, f_1 as offset
unbaseline	left	r	undo left baseline correction, inputting f_0
	right		undo right baseline correction, inputting f_0
	ends		undo ends baseline correction, inputting f_0 and f_1
mask	notch	r	mask with notches at f_0,f_2,…, with widths f_1,f_3…
	gauss		mask with gaussian at 0 with std. dev. f_0
	highpass		mask with trapezoid with cutoff f_0, cutoff width f_1
	lowpass		mask with trapezoid with cutoff f_0, cutoff width f_1
filter	notch	r	fourier filter with notch mask
	gauss		fourier filter with gauss mask
	highpass		fourier filter with highpass mask
	lowpass		fourier filter with lowpass mask

Possible error codes

1 out of memory
6 missing inputs to function
7 command not recognized
8 complex spectra not supported for this function
9 divide by zero
10 only one spectrum is complex
11 incompatible spectra x values
12 real spectra not supported for this function
13 fn2 not recognized
14 num value out of bounds

new creates a new real spectrum. copy copies a subset of a spectrum, between values f_0 and f_1, using either the original data points if $f_2 \leq 0$ or data points evenly spaced with spacing f_2. zerofill makes a new spectrum whose domain is f_0 times as large as the original, filling the extra space with zeros. If f_0 is negative, the new spectrum has the next larger integer power of 2 data points, otherwise f_0 needs to be at least 1. baseline returns a baseline corrected spectrum. unbaseline is the opposite of baseline, taking in the same inputs that baseline made as outputs.