SPIRE-UOL-DOC-001825

User Manual

for the

Data Processing Toolkit

for the

SPIRE test facility FTS

Input provided by Ian Chapman, Kris Dyke, Trevor Fulton, John Lindner, Locke Spencer
Assembled by Peter Davis, SPIRE local project manager, Canada
Approved by David Naylor, SPIRE Co-Investigator

Lethbridge, Canada
September 29, 2003

Table of Contents:

Scope of this document ... 3
Commonalities between all methods ... 3
Overview of all methods ... 4
apodize ... 4
checkLinearity ... 6
checkNoise ... 9
checkZpdQuality ... 11
coaddSpectra ... 12
correctPhase ... 14
deGlitch ... 16
detectGlitch ... 18
determineSampInterval ... 21
determineZPD ... 23
fourierTransform ... 25
getResponse ... 27
interpolate ... 29
ratioSpectra ... 32
subtractSpectra ... 34
TK_serializeSpectra ... 36
TK_createRawDataFromFits ... 38
Appendix ... 39

Table of Figures:

Figure 1: A sample double-sided interferogram before and after apodizing ... 5
Figure 2: Sample functions for the linear phase correction of an interferogram ... 15
Figure 3: A double-sided interferogram before and after deglitching ... 17
Figure 4: Sample data and threshold function ... 19
Figure 5: A parabola is defined by three points. It's highest point indicates ZPD ... 24
Figure 6: A sample interpolation using cubic spline ... 31

SCOPE OF THIS DOCUMENT

The User Manual provides users of the data processing toolkit with comprehensive help on the methods included in the toolkit. The current version aims to enable users to effectively analyze data collected with the test facility FTS. While the user manual assumes familiarity with the IDL© programming language, it pays particular attention to troubleshooting problems encountered while using the data processing toolkit.

More information on the data processing toolkit is available in the Design Document. It contains details on the following aspects of the data processing toolkit:

- file I/O
- class structure
- coding conventions
- exception handling
- details on the object structure, including limitations of the individual modules
- extension of the toolkit

The User Manual follows the structure of the methods of the data processing toolkit. All methods are listed alphabetically and discussed in detail.

COMMONALITIES BETWEEN ALL METHODS

Common to all methods are three keywords which are related to error-handling and troubleshooting:

> TK_ERROR: Set this keyword to a named variable that will contain the numerical error code should a run-time error occur during execution of this method. A complete list of error codes is included in the Appendix.
>
> TK_ERRMSG: Set this keyword to a named variable that will contain a string describing the run-time error should one occur during execution of this method
>
> DEBUG: Set this keyword to 0 for maximum performance. Set this keyword to 1 to see informational and error messages on the console. Set this keyword to 2 to add windows with graphical information on the data processing to the console-messages.

OVERVIEW OF ALL METHODS

apodize

Overview:

Apodize applies the user-specified apodization function to the double-sided subset of the interferogram object's signal array.

Calling Sequence:

apodizedArray=myInterferogram->apodize(inputArray, APOD_TYPE = [aNB_W_120 | aNB_M_140 | aP_141 | aHM_150 | aNB_S_160 | aD_174 | aBH_3_184 | aE_195 | aBH_4_221 | aBH_M4_222])

Arguments:

inputArray: The double-sided subset of the interferogram object's signal array. This argument is required and is assumed to be 3-Dimensional.

Keywords:

APOD_TYPE: Set this keyword to a string denoting the apodization function to be performed on the inputArray. The following values for this keyword are available:

1) aNB_W_120: Norton-Beer Weak
2) aNB_M_140: Norton-Beer Medium (default)
3) aP_141: P with $\alpha = -0.0325$ and $p = 0.3$
4) aHM_150: Hamming
5) aNB_S_160: Norton-Beer Strong
6) aD_174: D with $\alpha = 0.26$
7) aBH_3_184: Blackman-Harris 3 Terms
8) aE_195: E with $\alpha = 0.22$
9) aBH_4_221: Blackman-Harris 4 Terms
10) aBH_M4_222: Modified Blackman-Harris 4-terms

Output:

Returns the input signal array multiplied by the selected apodization function.

Optional Output:

None.

Detailed Description:

This method multiplies the selected apodization function point by point to each interferogram of the signal array passed as the first argument.

Restrictions & Error Codes:

Requires the IDL functions isEven() and CMREPLICATE().

The signal array is assumed to be three-dimensional ([row, column, signal]. To use this method properly, the signal array should contain double-sided interferograms, though this method does not check to see whether this is the case.

The output of this method is slightly different depending on whether the inputArray argument contains an even or odd number of elements. This stems from the fact that the apodization functions contain a unique central maximum. For the case where inputArray contains an odd number of elements, the size of apodization function is identical to that of the inputArray. When the number of elements in inputArray is even, the apodization function will also contain an even number of elements but its first element will be set to zero.

-401: The inputArray argument is not set.

-402: The inputArray argument is not a 3-D array.

-403: An unknown APOD_TYPE keyword has been specified. No apodization has been performed.

-426: The APOD_TYPE keyword not set. Using default (Norton-Beer Medium, aNB_M_140)

Example:

```
; Apply the 3-term Blackman-Harris apodization function to the double-sided
; portion of myInterferogram object's signal array.

; extract the double-sided portion from the myInterferogram's signal arrays
;
wh_ds=WHERE(*myIfgm.position LT myIfgm.ZPD, wh_count)
inputArray = (*myIfgm.signal)[*, *, 0:2*wh_count+1)

; Apply the 3-term Blackman-Harris apodization function
; Set the DEBUG flag to plot the results
apodizedArray = myIfgm -> apodize(inputArray, APOD_TYPE = aB_H_3_184,
/DEBUG)
```

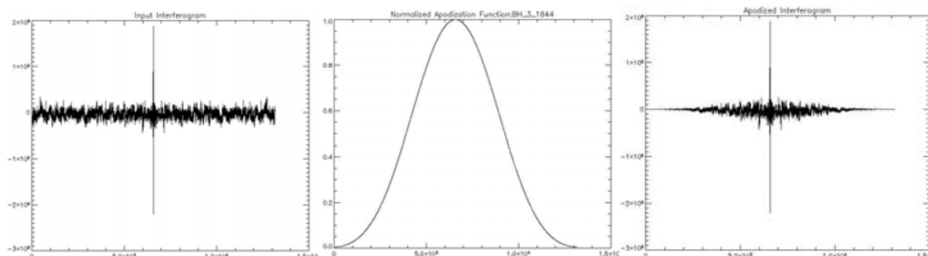


Figure 1: A sample double-sided interferogram before and after apodizing

Version History:

Version 1: 29 September 2003

checkLinearity

Overview:

CheckLinearity verifies the linearity of the stage Time array corresponding to the points where the stage position is sampled at given intervals.

Calling Sequence:

linearQuality = myRawData->checkLinearity([ACCELPOINTS=accelpoints] [, STAGEACCEL=stageaccel] [, ACCELTIME=acceltime] [, ACCELDIST=acceldist] [, LINEARTHRESHOLD=linearthreshold] [, CHISQ=chisq] [, NONLINARRAY=nonlinarray] [, DECELL=decell])

Arguments:

None.

Keywords:

LINEARTHRESHOLD: Defines the allowable deviation from the linear best fit. There is a default value of 20. The units are the same as the time array units (ticks from the 312500 Hz counter clock) and the type is ULONG.

DECELL: This binary keyword is used to truncate both ends of the time array to allow for stage acceleration and deceleration within the linearity analysis. If this keyword is not set, then only the acceleration phase truncated for the analysis. The same number of points are removed from the end of the interferograms as are removed from the initial acceleration phase. This truncation will not affect the array, it is only done for the linear regression, and no changes are made to the raw data object.

Output:

The method returns the ratio of good points over total points in the linear region of the stageTime array.A value identical or close to 1 indicates a high degree of linearity in the stage behaviour.

Optional Output:

ACCELPOINTS: Optional Keyword integer output indicating the number of points in the interferograms that are taken while the stage is accelerating and thus cannot be linear.

STAGEACCEL: Optional output of the acceleration of the stage assuming constant acceleration during the acceleration phase. Type is float and units are m/s2

ACCELTIME: Optional output of the time the stage is accelerating. Type is float and units are seconds.

ACCELDIST: Optional output of the distance the stage is accelerating in the initial acceleration phase. Type is float and units are microns.

CHISQ: Float type number representing chi-squared test output of linear fit of the linear portion of the time array. This output is taken directly from the LINFIT function.

NONLINARRAY: Array of integers indicating the points within the time array that are nonlinear. This only includes nonlinear points within the linear region of the array.

Detailed Description:

Discrete derivatives of velocity and acceleration are used to obtain keyword outputs STAGEACCEL, ACCELTIME, ACCELDIST, and ACCELPOINTS. A linear regression is performed on the time and position array. The difference of the time array with the best fit is analyzed using the input linear threshold to determine where the array is not linear. The array is then truncated with the initial acceleration phase removed, and the final deceleration phase is removed if the DECELL keyword is set. Another linear regression is performed after this truncation to get the data for the keywords CHISQ, and NONLINARRAY. The output is the ratio of good points within the linear region over the total points within the linear region. The object attributes are not changed by this method, the truncation mentioned is only local, not global.

Restrictions & Error Codes:

The Linearthreshold Keyword is VERY equipment sensitive. The default value was determined with a specific set of equipment but this may not be a good default value for other setups.

-176 The NonlinArray keyword output may be invalid as all data points are identified as beyond the threshold to test for linear stage behaviour.

-177 The stageAccel keyword output may be invalid as all data points fall within the threshold to test for linear stage behaviour.

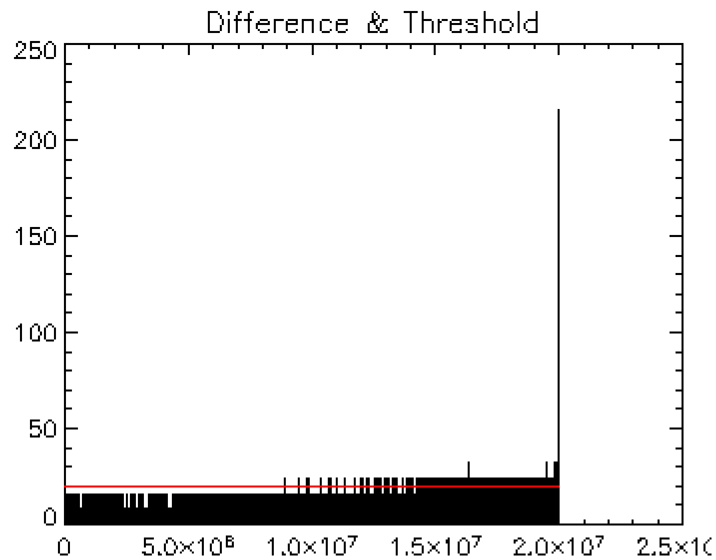
-178 Since the Decell keyword would have caused the method to truncate all data, the method proceeded as if the DECELL keyword was not set.

Example:

For the following input,

```
linearQuality = myRawData->checkLinearity( LINEARTHRESHOLD=20)
```

The ratio of good to total points in the linear region is output. The variation from linearity is plotted along with the linearThreshold value for a sample data set below.



The blue line represents the linearThreshold of 20 is was input in the calling sequence. There is no initial acceleration phase as all differences are below threshold at the beginning region. There are some nonlinear points after about the half way point. The output is the ratio of linear points to total points in the plot above.

Version History:

Version 1: 29 September 2003

checkNoise

Overview:

CheckNoise integrates the signal strength over three user specified bands of the power spectrum to give an indication of the quality of the transmission spectrum and the associated noise.

Calling Sequence:

noiseQuality = Spectrum->checkNoise(lowerBand, midBand, upperBand, width)

Arguments:

lowerBand:	Lower frequency band center units are cm-1.
midBand:	Frequency band center for portion within signal band.
upperBand:	Upper frequency band center.
width:	Width of each band to be integrated, units are cm-1.

Keywords:

None.

Detailed Description:

The arguments are checked to make sure they were entered in the proper order (low, med., high). The integration ranges are checked to make sure they are all within zero and the nyquist frequency. Ranges will be truncated to remain in this range if necessary. Integrations are performed over the specified regions and the output is returned.

Output:

The Vector noiseQuality contains three numbers for each pixel in the detector array. These numbers are the lower frequency range integration value, the mid band frequency range integration value, and the high frequency range integration value in that order.

Optional Output:

There are no optional outputs.

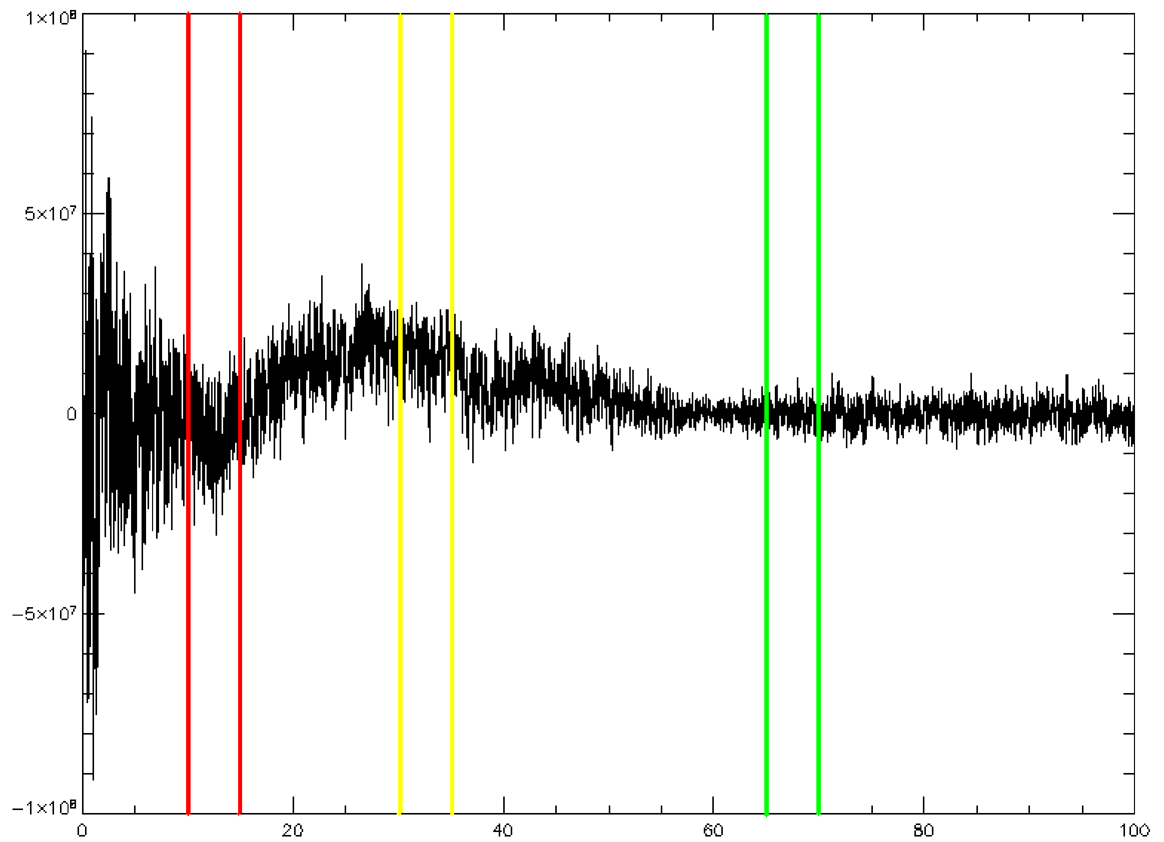
Restrictions & Error Codes:

None.

Example:

noiseQuality = Spectrum->checkNoise(12.5, 32.5, 67.5, 5)

The picture below shows a sample set of data for the checkNoise function. Each of the three highlighted regions below is integrated and the value is output for review/analysis by the user. The first three numbers input in the calling sequence above represent band centres for the low (red), mid (yellow), and upper (green) bands respectively. The next value is the integration width. All values are in units of wavenumbers (cm^{-1}). Each region is 5 wavenumbers wide, with the centre at the input location.



Version History:

Version 1: 29 September 2003

checkZpdQuality

Overview:

CheckZpdQuality determines the difference between the observedZpd and the expectedZpd.

Calling Sequence:

zpdQuality=myInterferogram-> checkZpdQuality ([QFLAG = qflag] [, SIGMA = sigma,])

Arguments:

None.

Keywords:

SIGMA: a double precision number to indicate how many standard deviations ZPDQUALITY can deviate from the mean before it is flagged in QFLAG. The QFLAG keyword must be set for this to be checked (Default: 1).

Output:

ZPDQUALITY: a 2D unsigned long integer array (unidex units = 10nm), containing the difference between the expected ZPD and the observed ZPD.

Optional Output:

QFLAG: this 2D sized integer array contains -1 if ZPDQUALITY is more than SIGMA standard deviations away from the mean of ZPDQUALITY. SIGMA can be defined by the user. If ZPDQUALITY is within SIGMA standard deviations QFLAG is set to 0. If all elements are 0, QFLAG is a scalar set to 0.

Detailed Description:

checkZpdQuality determines the difference between the observedZpd in the interferogram object and the expectedZpd for the optical setup. A value for each interferogram is saved in ZPDQUALITY. The user may specify a quality flag QFLAG, where QFLAG = -1 if ZPDQUALITY for an interferogram is more than SIGMA standard deviations away from the mean ZPDQUALITY. SIGMA can be specified by the used.

Error Codes:

-252: One or more zpdQualities are too far from the average.

Sample Results:

With an expected ZPD of 3547000 (in unidex units), one would get:

observedZpd	zpdQuality
3443612	-103388
3413655	-133345

Version History:

Version 1: 29 September 2003

coaddSpectra

Overview:

CoaddSpectra coadds, i.e. averages, the given array of spectrum objects, returning the result as an aggregate spectrum object.

Calling Sequence:

myCoaddedSpectrum = mySpectrum->coaddSpecta(spectrumArray [,STDDEV=stddev] [,MAXIMUM=maximum] [,MINIMUM=minimum])

Arguments:

spectrumArray: An array of spectrum objects to be coadded.

Keywords:

None

Output:

Returns the resulting spectrum as an aggregate spectrum object. The wavenumber array is given in cm^-1 and the transmission is fractional transmission (from 0 to 1).

Optional Output:

STDDEV: Set this keyword to a named variable to have the module return an aggregate spectrum that holds the standard deviation of the transmission at each wavenumber.

MAXIMUM: Set this keyword to a named variable to have the module return an aggregate spectrum that holds the maximum transmission at each wavenumber.

MINIMUM: Set this keyword to a named variable to have the module return an aggregate spectrum that holds the minimum transmission at each wavenumber.

Detailed Description:

This method coadds, or averages, the input spectra. The coaddition is performed for each pixel separately by averaging the transmission measured at each wavenumber. The result is then returned as an aggregate spectrum object. By using the optional keywords, the module will return additional spectrum objects containing the standard deviation, the minimum value, or the maximum value at each wavenumber for the input spectra.

Restrictions & Error Codes:

The spectrum array must contain more than a single spectrum. An array containing only one spectrum will cause an error to occur and for the original spectrum to be returned unchanged.

All spectra in the input array must have the exact same wavenumber array in order for the averaging to work. If any differences are detected, a fatal error is caused.

Results are best if the environmental conditions are steady across the spectrum array, but no error is thrown if the metadata is not identical across the whole array.

CoaddSpectra accepts both regular spectra and aggregate spectra as inputs. However, no weighting is used in the coaddition, so mixing of objects should be done with caution.

-3: Spectrum wavenumber arrays do not match across the objects in the array.

-776: At least 2 spectrum objects required as input. Returning input.

Example:

Coadd an array of spectra and return the standard deviation in another variable (ST_DV):

```
coaddedSpectrum = mySpectrum->coaddSpectra(spectrumArray, STDDEV=ST_DV)
```

Version History:

Version 1: 29 September 2003

correctPhase

Overview:

CorrectPhase corrects for a linear phase shift in an interferogram using the Forman method (see e.g. Griffiths, P. R. *Chemical Infrared Fourier Transform Spectroscopy*. New York: Wiley, 1975, 93-119).

Calling Sequence:

MyInterferogram->correctPhase (lowerBand, upperBand [, PCF_WID = pcf_wid] [, APOD_PCF = apod_pcf] [, APOD_DS = apod_ds])

Arguments:

LOWERBAND: a floating point scalar defining the lower pass band of the system (in cm^{-1}).

UPPERBAND: a floating point scalar defining the upper pass band of the system (in cm^{-1}).

Keywords:

APOD_DS: a string specifying the type of apodization function to apply to the double-sided interferogram. The aNB_W_120 is used as default.

PCF_WID: an unsigned long integer defining the width of the phase correction function (PCF) in number of points. As default, the maximum double-sided interferogram is used as basis for the PCF.

APOD_PCF: a string specifying the type of apodization function to apply to the PCF. The aNB_W_120 is used as default.

Output:

None.

Optional Output:

None.

Detailed Description:

correctPhase uses the Forman Method to correct for the linear phase shifts. The double-sided section of the interferogram is extracted from the signal array. It is apodized using the function type specified by APOD_DS. It is shifted so that ZPD is the first point and then Fast-Fourier Transformed (FFTed). The phase of the interferogram is determined by computing the arcTan of the ratio of the imaginary and real parts of the FFTed double-sided interferogram. A straight line is fit in the region of the passband (defined by LOWERBAND and UPPERBAND). This new phase is turned into a complex number as a function of wavenumber. This complex array is inverse-FFTed to return it to the spatial domain. The resulting function is shifted to bring the maximum point to the center and defines the PCF. It is apodized using the apodization type specified by APOD_PCF and then truncated to be the width specified by PCF_WID. This truncated apodizedPCF is convolved with the original interferogram to correct for the phase errors.

Restrictions & Error Codes:

-627: WHERE function failed to find indices in wavenumber array between the lowerBand and the upperBand.

Example:

The function call MyInterferogram->correctPhase (14.9, 50, PCF_WID=2800) will perform a linear phase-correction on an interferogram.

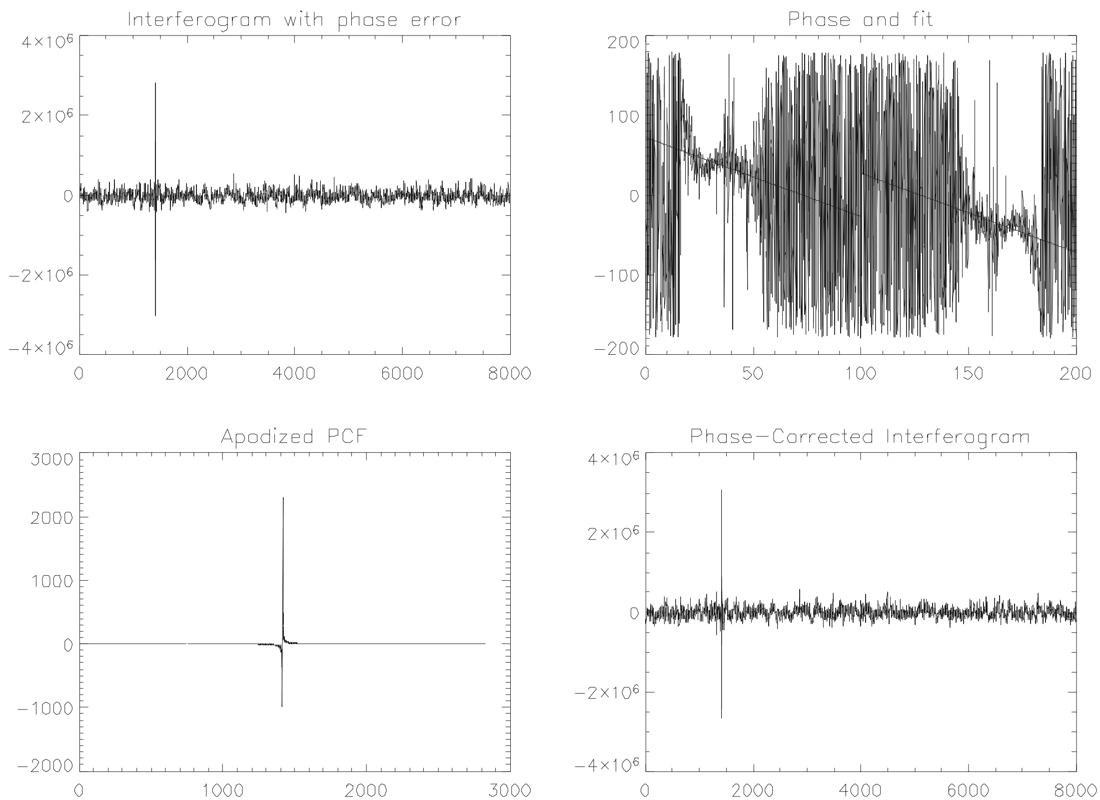


Figure 2: Sample functions for the linear phase correction of an interferogram

Version History:

Version 1: 29 September 2003

deGlitch

Overview:

This method removes spikes, typically due to cosmic rays, from the raw data's signal array. It identifies spikes with detectGlitch and replaces it with a fitted, straight line.

Calling Sequence:

MyRawData->deGlitch([ENVELOPE = envFct] [, INOFFSET=inOffset] [, OUTOFFSET=outOffset] [, FWHM=fwhm] [, INOUT=inout] [, BADPTS=badpts] [, QFLAG=qflag])

Arguments:

None.

Keywords:

ENVELOPE: used and explained in detectGlitch

INOFFSET: used and explained in detectGlitch

OUTOFFSET: used and explained in detectGlitch

FWHM: used and explained in detectGlitch

INOUT: used and explained in detectGlitch

BADPTS: used and explained in detectGlitch

Detailed Description:

deGlitch is a nested module that calls detectGlitch; it will not work without detectGlitch. If the user wants to pass any keywords in detectGlitch, they must be passed when deGlitch is called. deGlitch runs detectGlitch to find any spikes. deGlitch removes those spikes by fitting a straight line through the points that have spikes and replacing those spikes with the value from the linear line.

Output:

None.

Optional Output:

QFLAG: a 2D array of integers, set to -1 when more points than BADPTS are above the threshold for any given interferogram. Otherwise, values are set to zero.

Restrictions & Error Codes:

-377 DeGlitch failed to remove all spikes.

-378 If there is a spike in the first or last index of any signal array, deGlitch cannot correct it since the fitting routine uses the signal around the position of the spike to compute the fit.

Example:

MyRawData->deGlitch(ENVELOPE=eLorentz , INOFFSET=20, OUTOFFSET=15 , FWHM=10, INOUT=200, BADPTS=1, QFLAG=qflag)

Figure 3 shows the before and after image of an interferogram with a large spike (compare Figure 4). In Figure 3a, a spike is clearly visible. After detection with detectGlitch, deGlitch removes the spike and replaces it with a straight line, as seen in Figure 3b.

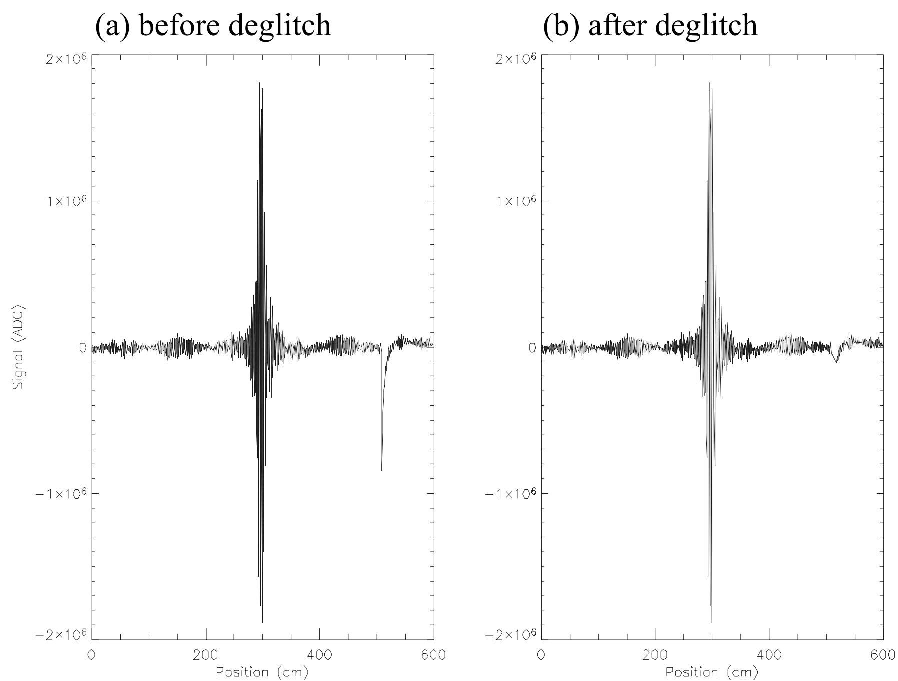


Figure 3: A double-sided interferogram before and after deglitching

Version History:

Version 1: 29 September 2003

detectGlitch

Overview:

DetectGlitch identifies spikes in the signal data (typically due to cosmic rays). Any signal that is above a user-defined threshold is considered a spike.

Calling Sequence:

identifyGlitch = myRawData-> detectGlitch([ENVELOPE = envFct] [, INOFFSET=inOffset] [, OUTOFFSET=outOffset] [, FWHM=fwhm] [, INOUT=inout] [, BADPTS=badpts] [, QFLAG=qflag])

Arguments:

None

Keywords:

ENVELOPE: a string corresponding to the type of function used to compute the threshold. Available types: ‘eLorentz’, ‘eGauss’ (Default: ‘eLorentz’).

INOFFSET: an integer specifying the percentage of scaling to offset the inner points by (Default: 20%)

OUTOFFSET: an integer specifying the percentage of scaling to offset the outer points by (Default: 15%).

FWHM: an integer specifying the percentage of the width of the interferogram as the full-width half-maximum (Default: 10%).

INOUT: an integer specifying the number of points for which INOFFSET extends (Default: 200).

BADPTS: the integer number of bad points (those above the threshold) acceptable; QFLAG keyword must be set for this to be checked (Default: 20)

Figure 4: Sample data and threshold function

Output:

IDENTIFYGLITCH: a 3D integer array the same size as the signal array. Any index marked with a 1 indicates a spike in the signal data; any index with a 0 indicates no spike (i.e. if IDENTIFYGLITCH[0,1,472] = 1, then the interferogram at row 1, column 2 has a spike at index 473).

Optional Output:

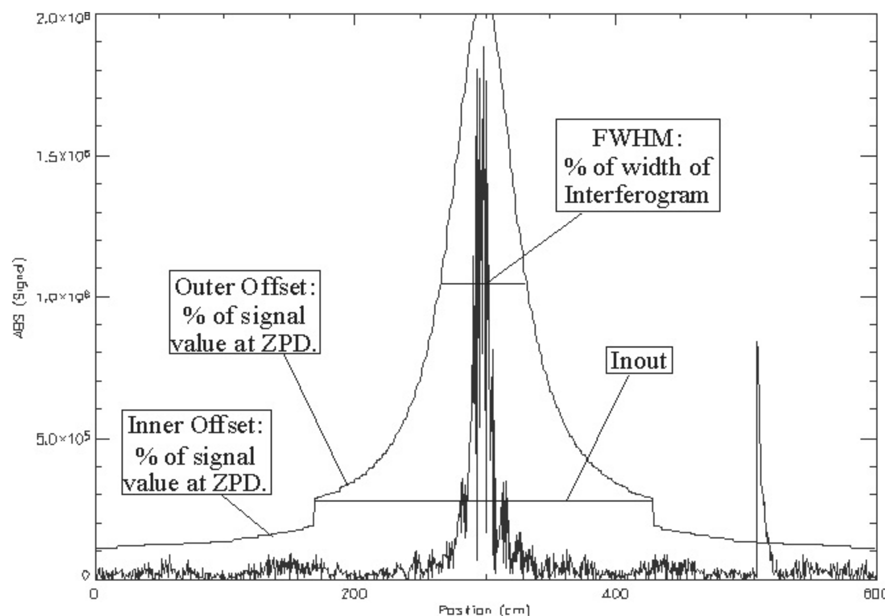
QFLAG: a 2D array of integers, set to -1 when more points than BADPTS are above the threshold for any given interferogram. Otherwise, values are set to zero.

Detailed Description:

DetectGlitch defines a threshold for each interferogram. It compares the absolute value of the interferogram to the threshold. Any point of the absolute interferogram above the value of the threshold at the same point is considered a spike. A ‘1’ is then marked in a 3D array called IDENTIFYGLITCH which is the same size as the bolometer signal array. If no spike is found, a ‘0’ is marked in IDENTIFYGLITCH.

The user can pass up to five parameters that define the shape of the threshold: INOFFSET, OUTOFFSET, INOUT, FWHM, ENVELOPE, discussed above.

The user may set a quality flag QFLAG, such that if, for a given interferogram, there are more points above the threshold than is acceptable, QFLAG is marked as -1. What are an acceptable number of points above the threshold is set by another keyword, called BADPTS. detectGlitch can be run on its own. It is called from



deGlitch to identify the spikes to be deglitched.

Restrictions & Error Codes:

No module-specific errors.

Example:

Figure 4 shows the absolute value of an interferogram. The threshold function is overplotted. As can be seen, there is a spike at around 500.

Glitches=myRawData->detectGlitch() yields the following result for the data shown above:

Index i	…	3999	4000	4001	4002	4003	4004	4005	…
Glitches[0,0,i]	…	0	1	1	1	1	1	0	…

Version History:

Version 1: 29 September 2003

determineSampInterval

Overview:

DetermineSampInterval determines the sampling interval for raw data based on the corresponding position array.

Calling Sequence:

myRawData->determineSampInterval([MDEV=mdev] [, SDEV=sdev] [, VARIANCE=variance] [, SKEWNESS=skewness] [, KURTOSIS=kurtosis])

Arguments:

None

Keywords:

None

Output:

None, the RawData attribute sampInteval is set.

Optional Output:

MDEV: The Mean deviation of points from the average, just as in IDL'sMOMENT function.

SDEV: Standard Deviation as in IDL's MOMENT function.

VARIANCE: Variance as in IDL's MOMENT function.

SKEWNESS: Skewness as in IDL's MOMENT function.

KURTOSIS: Kurtosis as in IDL's MOMENT function.

Detailed Description:

The position array is shifted by one and subtracted from the original to obtain an array containing all of the sampling intervals between interferogram sample points. The difference array is then passed to the IDL MOMENT function to obtain the average, along with any desired optional keyword outputs (as listed above). A device based scaling factor is then multiplied with the average geometric sampling interval to arrive at the optical sampling interval. This scaling factor comes from a "getOPDfactor" method, for a Michelson Interferometer this factor is 2.

Restrictions & Error Codes:

The result obtained is not compared to the expected sampling interval (the input value from the operator). The accuracy of the sampling interval is limited to one Unidex unit.

-201 Sampling Interval was less than one unidex unit, there is likely a problem with the stagePosition array

-202 Sampling Interval is greater than the width of the stage, there is likely a problem with the stagePosition array

Example:

myRawData->determineSampInterval() sets the sampling interval in the metaAssistant. DeltaX=MyRawData->getSampInterval() returns 5000 for a 50 μm sampling interval.

Version History:

Version 1: 29 September 2003

determineZPD

Overview:

DetermioneZPD determines ZPD from an interferogram. It looks for the highest peak of the signal array. Then, using a fitting routine, the exact position value of the zpd is determined.

Calling Sequence:

MyInterferogram->determineZPD

Arguments:

None.

Keywords:

None.

Output:

None.

Optional Output:

None.

Detailed Description:

determineZPD first determines where the peak of the interferogram is. It takes a point on either side of the peak and calculates a 2nd order polynomial fit to those three points. It then determines what the maximum point of the fit is and finds the corresponding stage position corresponding to it. This is then set as the observedZpd attribute for that interferogram object..

Restrictions & Error Codes:

-478: ZPD appears to be at the 0'th position of the interferogram.

Sample Results:

The maximum value is 1805534, at a position of 294.13. On either side, the values are 65165 at 293.48 and 539059 at 294.78. In Figure 5, the peak of the quadratic is slightly to the right of the point at 294.13. The procedure call MyInterferogram->determineZPD sets observedZpd to a value of 294.179.

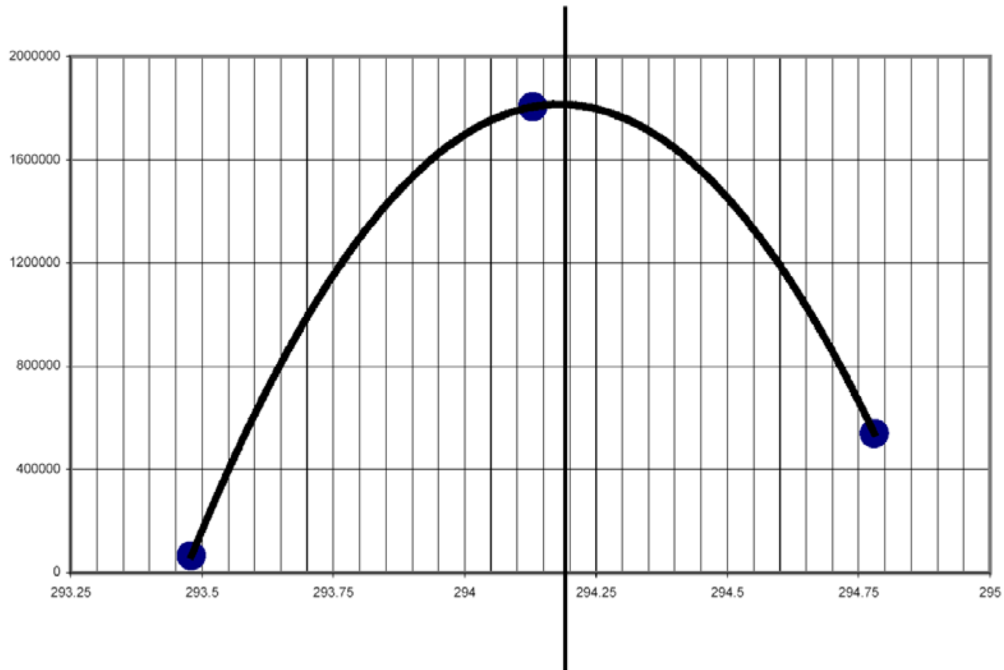


Figure 5: A parabola is defined by three points. It's highest point indicates ZPD

Version History:

Version 1: 29 September 2003

fourierTransform

Overview:

FourierTransform performs a Fourier Transformation (FT) on an interferogram object, returning the result as a spectrum object.

Calling Sequence:

mySpectrum = myInterferogram->fourierTransform([/DS] [, /ZEROPAD] [, APOD=apod])

Arguments:

None.

Keywords:

DS: Set this keyword if the interferogram is double sided, i.e. has ZPD at its centre. The module then uses the appropriate algorithm to perform the FT.

ZEROPAD: Set this keyword to have the interferogram padded with zeroes to extend its length to 2^n automatically.

APOD: Set this keyword to a string indicating the apodization function to use in the FT (see apodize for further details).

Output:

Returns the Fourier-Transformed interferogram as a spectrum object. Wavenumber is given in cm^-1 and Transmission is the fractional transmission (from 0 to 1).

Optional Output:

None.

Detailed Description:

The fourierTransform module performs an FT on the interferogram that calls the method. Each pixel's interferogram is transformed separately using the FFT routine built in to IDL. The module accepts both single-sided and double-sided interferograms (with the use of a keyword). The interferogram can be zero–padded through use of an optional keyword, extending the length of the data array to 2^n (increasing the spectrum resolution and the efficiency of the FT routine). The value of "n" is chosen to make the interferogram approximately 4 times longer than the original. Apodization functions can be used through the use of the APOD keyword. Set this keyword to the name of the apodization function that you wish to use (see the Apodization documentation for a list of available functions). The output of this module is a spectrum object.

Restrictions & Error Codes:

The input of this module is restricted to single-sided and double-sided interferograms. These must be input in ascending wavenumber order. A double-sided interferogram MUST be indicated by setting the DS keyword in the function

call. Not setting the keyword will cause a double-sided interferogram to be analyzed as though it were single-sided.

-551: The keyword for processing a double-sided interferogram was set when processing a single-sided interferogram.

Example:

Perform an FT on a double-sided interferogram with Norton-Beer Weak Apodization:

mySpectrum = myInterferogram->fourierTransform(/DS,APOD = aNB_W_120)

Version History:

Version 1: 29 September 2003

getResponse

Overview:

GetResponse calculates the spectral response of the bolometers using spectra collected with a "hot" blackbody and a "cold" blackbody

Calling Sequence:

Response = myCommonAssistant->getResponse(hotSpectrum,coldSpectrum)

Arguments:

hotSpectrum: A spectrum or aggregate spectrum object collected with the blackbody operating at its warmer temperature.

coldSpectrum: A spectrum or aggregate spectrum object collected with the blackbody operating at its cooler temperature.

Keywords:

None.

Output:

Returns the spectral response as a float array. The units are (W^-1 * steradian * cm^-1)

Optional Output:

None.

Detailed Description:

The spectral response is calculated for each pixel separately. Using a radiative transfer model (I.M. Chapman, M.Sc. Thesis, 2002, University of Lethbridge), the atmospheric transmission is simulated for the environmental conditions during the tests and is used to estimate the radiance that should reach the detector. The response is calculated by dividing the difference in the measured signal between the "hot" and "cold" scans by the difference between the spectral radiance of the blackbody operating at the two temperatures modified by the atmospheric transmission.

Restrictions & Error Codes:

The two spectra in the input array must have the exact same wavenumber array in order for the ratio to work. If any differences are detected, a fatal error is caused. The order of input is important since the analysis will be incorrect if the "hot" and "cold" spectra are mixed up.

-3: Spectrum wavenumber arrays do not match across the objects in the array.

-61: Temperature out of range in TK_interpolateCube.

-62: Pressure out of range in TK_interpolateCube.

-63: Humidity out of range in TK_interpolateCube.

-64: Out of range exponent value in Planck equation in TK_Blackbody, i.e. black body temperature too hot.

Example:

Calculate the spectral response:

Response = myEnviroCharacteristics -> getResponse(hotSpectrum, coldSpectrum)

Version History:

Version 1: 29 September 2003

interpolate

Overview:

This method creates a new interferogram object by relating the signal data from the bolometers to the position data from the linear stage. It interpolates the bolometerSignal data from the times in the bolometerTime array to those in the stageTime array.

Calling Sequence:

oInterferogram = myRawData->interpolate([INTERPOL_TYPE = iSPLINE | iLAGRANGE | iCHEBY | iLINEAR])

Arguments:

None.

Keywords:

The type of interpolation to be performed can be selected with a keyword:

iSPLINE: Set this keyword to perform cubic spline interpolation. (Default)

iLAGRANGE: Set this keyword to perform interpolation using Lagrange basis functions.

iCHEBY: Set this keyword to perform interpolation using Chebyshev basis functions.

iLINEAR: Set this keyword to perform linear interpolation.

Output:

oInterferogram: An interferogram object containing the interpolated bolometer signal as a function of stage position. The bolometer signal is a floating point 3-D array while the stage position is a 1-D array of unsigned long (32-bit) integers.

Optional Output:

None.

Detailed Description:

This method interpolates the RawData object's bolometerSignal array data attribute onto the stagePosition grid via bolometerTime and stageTime. As a result the bolometerSignal array becomes a function of stagePosition. These arrays are then used (along with the metadata associated with the RawData object) to create a new interferogram object.

The default interpolation method is Cubic Spline. Cubic Spline interpolation was selected after tests performed by the AIG group at the University of Lethbridge where the interpolation with cubic spline has proven to be ten times better than a linear interpolation for the intended application.

The interpolated signal array that is part of the new interferogram object is truncated to that portion of the stageTime array that was encompassed by the values in the bolometerTime array.

Some extra points at the beginning and end of the interpolated signal may also be truncated depending on the type of interpolation function used. The number of truncated points is as follows:

SPLINE: 2 points at the beginning and end.

LAGRANGE: 2 points at the beginning and end.

CHEBYSHEV: 5 points at the beginning and end.

LINEAR: 1 point at the beginning and end.

Restrictions & Error Codes:

-101: There is not enough overlap between bolometerTime/stageTime data to perform an interpolation. This error indicates that the values of the bolometerTime data attribute of the RawData object do not overlap with those in the stageTime array. This makes interpolation impossible and means that an interferogram cannot be generated.

-102: INTERPOL_TYPE is set but is not equal to a valid value. The input is returned without any changes.

-126: The INTERPOL _TYPE keyword not set. Using default (SPLINE)

If more than one Keyword is specified, the function does NOT return an error. Rather, the highest-priority option is performed. Priority is given by the order that the Keywords are listed above.

Example:

```
>; Inspect the contents of a RawData object
>;
>bolmeterTime = my RawData->getBolometerTime()
>bolmeterSignal = my RawData->getBolometerSignal()
>stageTime = my RawData->getStageTime()
>stagePosition = my RawData->getStagePosition()
>PRINT, bolmeterTime
[0.000000      1.00000      2.00000      3.00000      4.00000      5.00000
6.00000      7.00000      8.00000      9.00000]
>PRINT, bolmeterSignal
[-50.0000     -32.0000     -18.0000     -8.00000     -2.00000     0.000000
-2.00000     -8.00000     -18.0000     -32.0000]
>PRINT, stageTime
[0.500000      1.50000      2.50000      3.50000      4.50000      5.50000
6.50000      7.50000      8.50000      9.50000      10.5000]
>PRINT, stagePosition
```

```
[200.000      201.000      202.000      203.000      204.000      205.000
206.000      207.000      208.000      209.000      210.000]
```

>; Create a new interferogram object using piecewise cubic spline interpolation

>;

> oInterferogram = myRawData -> interpolate(INTERPOL_TYPE=iSPLINE)

> signal = oInterferogram->GETsignal()

>PLOT, bolmeterTime, bolmeterSignal, PSYM = 4

>OPLOT, *myRawData.stageTime, signal, PSYM = 4

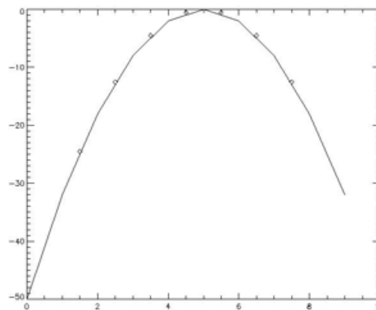


Figure 6: A sample interpolation using cubic spline

Version History:

Version 1: 29 September 2003

ratioSpectra

Overview:

RatioSpectra calculates the ratio between two spectra in an array, returning the result as an aggregate spectrum object.

Calling Sequence:

myRatioedSpectrum = mySpectrum -> ratioSpecta(spectrumArray [, MINIMUM=minimum] [, MAXIMUM=maximum])

Arguments:

spectrumArray: An array of two spectrum objects to be ratioed.

Keywords:

None

Output:

Returns the resulting spectrum as an aggregate spectrum object. The wavenumber array is given in cm^-1 and the transmission is fractional transmission (from 0 to 1).

Optional Output:

MAXIMUM: Set this keyword to a named varaiable to have the module return an aggregate spectrum that holds the maximum transmission at each wavenumber.

MINIMUM: Set this keyword to a named variable to have the module return an aggregate spectrum that holds the minimum transmission at each wavenumber.

Detailed Description:

The ratio is calculated for each pixel separately by dividing the first spectrum by the second in the array at each wavenumber (C=A/B). Minimum or maximum value spectra can be obtained using the optional keywords provided. The results are returned as aggregate spectrum objects.

Restrictions & Error Codes:

The two spectra in the input array must have the exact same wavenumber array in order for the ratio to work. If any differences are detected, a fatal error is caused. If more than two spectra are input, an error message is generated and the module calculates the ratio between the first two spectra only. If fewer than two spectra are input, a fatal error is generated.

-3: Spectrum wavenumber arrays do not match across the objects in the array.

-901: Input spectrum array contains fewer than 2 spectra.

-926: Input spectrum array contains more than 2 spectra. Only the first two spectra in the array are used.

Example:

Find the ratio between two spectra from different days and return the maximum value at each wavenumber to a spectrum name maxSpec:

ratioedSpectrum = mySpectrum -> ratioSpectra([day1Spectrum, day2Spectrum],MAXIMUM=maxSpec)

Version History:

Version 1: 29 September 2003

subtractSpectra

Overview:

SubtractSpectra calculates the difference between two spectra, returning the result as an aggregate spectrum object.

Calling Sequence:

mySubtractedSpectrum = mySpectrum->subtractSpecta(spectrumArray [, MINIMUM=minimum] [, MAXIMUM=maximum])

Arguments:

spectrumArray: An array of two spectra. The second in the array is subtracted from the first (C=A-B).

Keywords:

None

Output:

Returns the resulting spectrum as an aggregate spectrum object. The wavenumber array is given in cm^-1 and the transmission is fractional transmission (from 0 to 1).

Optional Output:

MAXIMUM: Set this keyword to a named variable to have the module return an aggregate spectrum that holds the maximum transmission at each wavenumber.

MINIMUM: Set this keyword to a named variable to have the module return an aggregate spectrum that holds the minimum transmission at each wavenumber.

Detailed Description:

The difference is calculated for each pixel separately by finding the transmission difference at each wavenumber. Minimum or maximum value spectra can be obtained using the optional keywords provided. The results are output as aggregate spectrum objects.

Restrictions & Error Codes:

The two spectra in the input array must have the exact same wavenumber array in order for the difference to work. If any differences are detected, a fatal error is caused. If more than two spectra are input, an error message is generated and the module calculates the difference between the first two spectra only. If fewer than two spectra are input, a fatal error is generated.

-3: Spectrum wavenumber arrays do not match across the objects in the array.

-801: Input spectrum array contains fewer than 2 spectra.

-826: Input spectrum array contains more than 2 spectra. Only the first two spectra in the array are used.

Example:

Find the difference between a hot and a cold spectrum and return a spectrum object named minArray, containing the minimum value at each wavenumber:

differenceSpectrum = mySpectrum -> subtractSpectra([hotSpectrum, coldSpectrum], MINIMUM=minArray)

Version History:

Version 1: 29 September 2003

TK_serializeSpectra

Overview:

serializeSpectra writes the array of spectrum objects, taken as input, to the fits interface, returning an array of filenames, one for each spectrum object.

CallingSequence:

spectrumFilenames = TK_serializeSpectra(oSpectrumArray, outputDir [,SERIALIZED])

Arguments:

oSpectrumArray: The array of spectrum objects that are to be written to fits files.

ouputDir: A string specifying where the output files should be written.

Keywords:

None

Output:

Returns a string array of filenames. If all objects serialized successfully then the number of elements in the output will be equal to the number of objects in the input array. If not all objects in the input array serialized successfully, the output from the SERIALIZED keyword can be used to deduce which files failed to serialize. For regular spectra the filename will be in the form: rSpectrum_obsid_bbid_date_x.fits. For aggregate spectra the filename will be in the form: aSpectrum_date_x.fits. Where x is an arbitrary integer used to ensure that for a given call to the serialize method there will be no file naming conflicts.

Optional Output:

SERIALIZED: Set this keyword to a named variable that upon return will contain an array of indices indicating which objects in the array were successfully serialized.

Detailed Description:

The TK_serializeSpectra first checks to ensure that all objects in the array are valid spectrum objects and removes any that are not. It then loops through the input array checking whether the object is a TK_RegSpectrum or TK_AggregateSpectrum. If the object is an aggregate spectrum it is serialized immediately. If the object is a regular spectrum it checks the next object in the array to see if it is from the same raw data input file. If it is not it serializes the object and those previous that belong in the same output file. Note that the method assumes that regular spectrum objects that should be written to the same output file are adjacent in the input array.

Restrictions & Error Codes:

The method assumes that regSpectrum that are supposed to be written to the same output file (i.e. they came from the same input file), are adjacent in the input array. If they are not, they will be written into a separate file.

-978: All objects in the input array were invalid, no files written

TK_createRawDataFromFits

Overview:

The TK_createRawDataFromFits routine is used to ingest the data from the HCSS database into the data processing toolkit. It takes an fts fits file and a bolometer fits file and returns an array of TK_RawData objects representing the 'half-scans' that were contained in the input files.

CallingSequence:

rawDataObjects = TK_CreateRawDataFromFits(ftsFitsFile, bolFitsFile, expZpd)

Arguments:

ftsFitsFile: A string indicating the filename of the fts fits file to be used to create the raw data objects

bolFitsFile: A string specifying the filename of the bolometer fits file to be used to creat the raw data objects. Note that the two input files must be taken from the same 'scan'.

Keywords:

None

Output:

Returns an array of TK_RawData objects, one RawData object for each 'half-scan'.

Optional Output:

None

Detailed Description:

The TK_CreateRawDataFromFits routine first reads in the fts file and parses out the appropriate data into 'half-scans'. Then for each half scan the max and min value of the dpu clock count is taken and used to parse out the half-scans from the data in the bolometer fits file. Note that the method will attempt to take 10 points (10 * 1562.5 dpu pulses) on either side of the fts min and max values in order to ensure overlap for the interpolation routine.

Restrictions & Error Codes:

The method will check to make sure that the observation ID and Building Block ID are the same for both files indicating that the files contain data from the same scan.

-53: Obsid and bbid of both input files should be equal (i.e. from the same scan)

APPENDIX