SPIRE User's Manual

version 1.3, Document Number: SPIRE-RAL-DOC 003231
30 July 2010

SPIRE User's Manual

Table of Contents

Preface

1. Versioning

On the front page of this manual is a version number made of three digits. The first two digits follow a traditional versioning system (0.1, 0.2, ...), and the changes introduced with each version are detailed below. The third digit is the SPIRE build number to which each edition of the manual is associated. Also shown on the front page is the date of publication of the manual.

1.1. Changelog

The following was changed for v1.3

- General updates to text and figures for engineering and spectroscopy pipeline modules. Added two new sections to spectrometer pipeline on the creation of a SPIRE spectral cube (SPIRE-1793).

The following was changed for v1.2

- Updated to adjust Check ADC flags and truncation module description.

The following was changed for v1.1

- Updated to adjust BSM error descriptions.

The following was changed for v1.0

- Updated to adjust Spire Pointing Product descriptions.

The following was changed for v0.9

- Spectrometer Pipeline Section Updated.

The following was changed for v0.8

- Spectrometer Pipeline Section Updated.

The following was changed for v0.7

- Spectrometer Pipeline Section Updated - adjusted wordings, changed import statements, parameter names and sample code to reflect the status at the end of the 1.2 branch

  Merged Spire Pointing Product chapters into photometery and spectroscopy chapters, as now a module..

The following was changed for v0.6

- Photometer Pipeline Section Updated - adjusted wordings, changed import statements, parameter names and sample code to reflect the status at the end of the 1.2 branch

  Auxillary products Section Added - added chapter on Spire Pointing Product.

The following was changed for v0.5

- Spectrometer Pipeline Section Updated - adjusted wordings, changed import statements, parameter names and sample code to reflect the status at the end of the 1.2 branch

The following was changed for v0.4

- SPIRE pipline section updated

- Flowcharts updated
- Photometer Pipeline Section Updated following Science Validation Review
- Spectrometer Pipeline Section Updated following Science Validation Review

The following was changed for v0.3

- Original UM was replaced with sections from the HOWTOs, plus the SPIRE Pipeline User's Guide, which will form the basis of the UM.

The following was changed for v0.2

- Added this preface.
- *Chapter 1*: Added workaround to reintroduce SPIRE specific help into QLA, including context-sensitive help.
- *Chapter 2*: Added the preexisting QLA manual as a new chapter.

The following was changed for v0.1

- First version of the manual, incudes the SPIRE DP Quick Setup guide as its only chapter.

Chapter 1. Introduction

1.1. Scope of this User's Manual

This document explains the SPIRE pipeline and its use for reduction of photometer and spectrometer data using the standard Observatory Functions (POF, SOF Astronomical Observing Templates, AOTs). This document is aimed at the astronomical user and calibration scientists who may wish to use the pipeline from a scientific viewpoint. This document is not meant for developers and those interested in the details of the pipeline module algorithms and processes.

This document provides the procedures to install and set up the data reduction environment (Herschel Common Science System, HCSS). A broad overview of the pipeline is given followed by detailed information of the individual pipeline steps including parameters and settings. The pipeline corresponding to each AOT is discussed. Worked examples are also provided.

1.2. The SPIRE Pipeline

The SPIRE pipeline is a part of the general Development Pipeline of the Herschel Common Science System (HCSS) being developed by the Herschel Science Center (HSC) and Herschel Instrument Control Centres (ICCs) to provide the complete software system for the Herschel Observatory mission. The entire Development Pipeline is written in Java and scripted in Jython and is a fully supported stand alone system. The pipeline can be run via scripts either end-to-end or stepwise. In addition to the standard product pipeline processing the Herschel Data Processing system will also provide the tools to interactively reduce the data through Graphical User Interfaces provided by the Herschel Interactive Processing Environment (HIPE) The pipeline supports processing for 6 different AOTs. In addition each AOT may have independent parameters to further refine the type of observation to be carried out.

An overview of the SPIRE pipline is shown in the figure below. The raw telemetry data has already been processed into the basic Level 0 Raw data format by a pre-processing step during the Operational Day Processing. The SPIRE pipeline as delivered to a user takes these Level 0 products as input. The Level 0 to Level 0.5 processing is referred to as the engineering conversion or "common pipeline" and takes the raw ADU in the timeline data and converts them to meaningful units (e.g. Volts, etc). The next step is the AOT specific processing (large map, small map, point source or spectrometer pipelines) which will produce calibrated timelines as Level 1 products. The advanced processing steps will take the Level 1 products and produce maps, spectral cubes and point source flux estimations. Quality control will be handelled by ESA rather than the ICC.

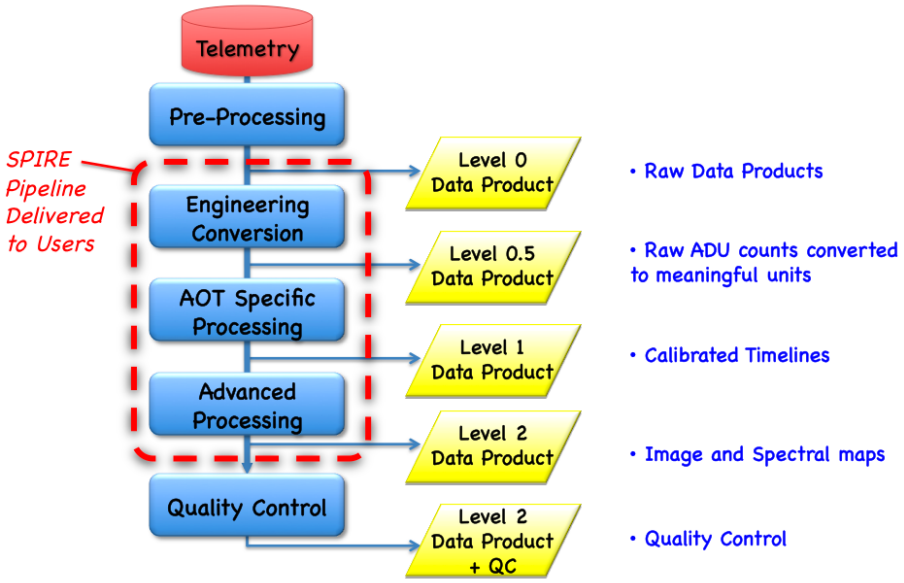

Figure 1.1. An overview of the SPIRE pipeline from the raw telemetry to the fully processed products.

1.3. Installation and Startup of HIPE

Data reduction for SPIRE datasets is performed using the Herschel Interactive Processing Environment (HIPE). HIPE is part of the Herschel Data Processing system and can be installed with the software installer (see Herschel Science Centre website). Software can be run on a server or individual workstation running Windows XP, Linux or Solaris. The minimum recommended system is Windows/Linux 32-bit w/1GB RAM or 64-bit W/Lin/Mac w/1GB RAM; Browsers for use with the systemm (including download) IE 6+ , Netscape 7+, Mozilla (Firefox) 1.5+, Safari (Mac). The system is Java based and requires Java 1.6. General Java scripts can be run on the system. Installation instructions are provided at the bottom of the FTP page.

Once the software is installed, HIPE can be started by several means. Using Windows, Herschel software can be started under the "Start" menu after a standard installation. Alternatively, HIPE can be started from a command line.

```
hipe
```

For more details on HIPE, please review the HIPE Introduction section in the Herschel DP HowTos docuument.

Chapter 2. Accessing the Raw Data

2.1. Accessing the Data

The Raw Data Extraction step is the first step in the SPIRE data processing pipeline. The purpose of this step is to extract the telemetry data (specified by a user selected range) from a database and to compile them into a set of Level-0 SPIRE data products for the use of further data processing steps.

In addition to producing the Level 0 products. It also produces the DPU reset history calibration product.

The Herschel Science Archive (HSA) is the main repository for the observational data products from Herschel. It is available via a web interface to the Herschel Science Centre. But it is also available directly from within a DP session using HIPE. For more information on acessing data sets via the HSA or through HIPE, please review the Herschel DP HowTo on accessing and retrieving data. This document is accessible through the Documentation button in HIPE.

Chapter 3. Processing the Raw Data: The Engineering Conversion

3.1. Introduction

This part of the SPIRE processing pipeline is common to both the photometer and spectrometer and converts the Level 0 raw data products into the processed Level 0.5 data products. The Pipeline to convert the raw data, i.e. the Level 0 Product, into the processed Level 0.5 Product is shown in Figure 3.1

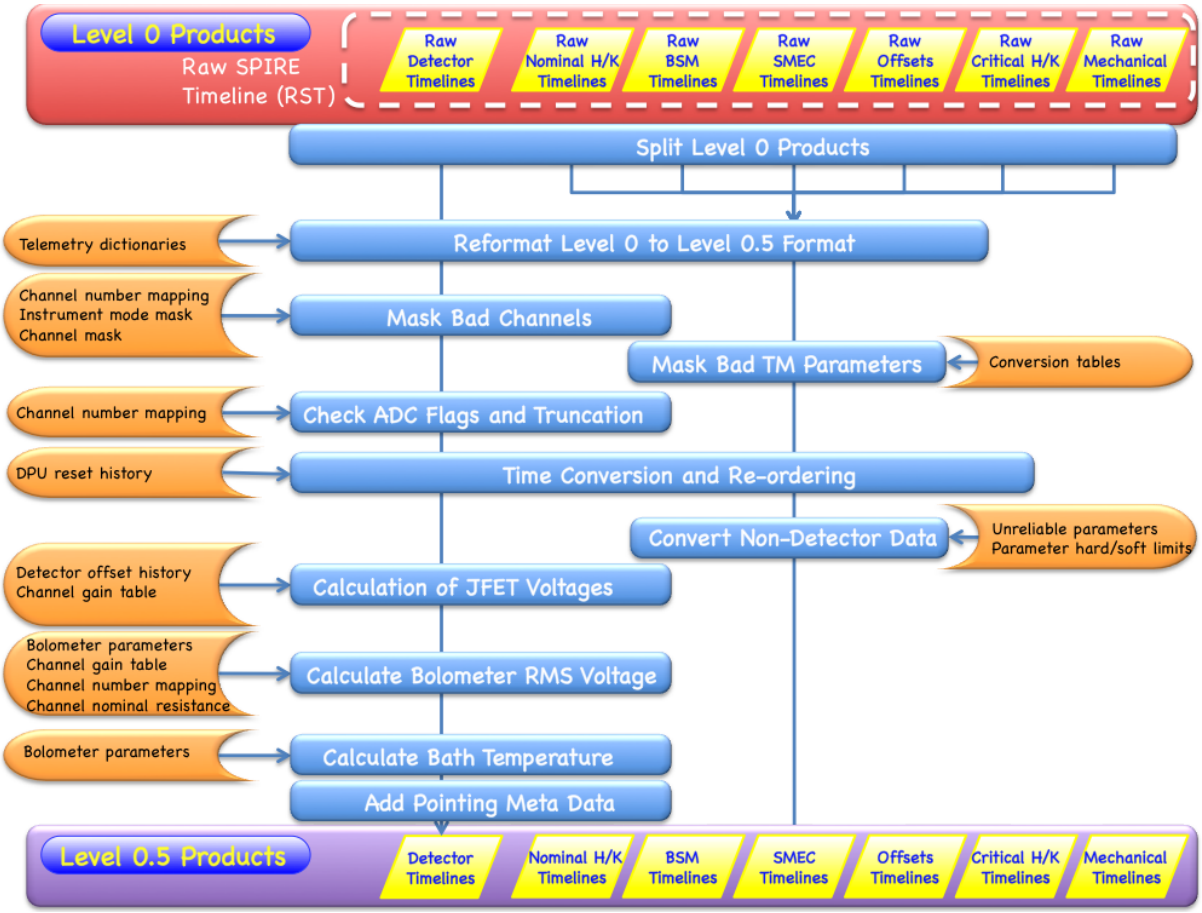

Figure 3.1. The Engineering Conversion Pipeline takes the raw Level 0 Products and converts the ADU counts into meaningful units (volts, etc).

This module is composed of several tasks. Each task can be executed as an independent step. The module provides also a main task, named `EngConversionTask`, that calls all the tasks of this module in the correct sequence.

The usage of the `EngConversionTask` instead of the single internal tasks is recomanded because the processing flowchart of this step is quite complex. Moreover, the internal tasks work only on single products while the `EngConversionTask` can process a complete set of products in one single execution.

3.2. Running the *Engineering Conversion* Pipeline

3.2.1. Module Description

The *Engineering Conversion* Pipeline is responsible for the processing of Level 0 products to Level 0.5 products.

3.2.2. Input Data Products

level0	Input Level-0 SPIRE Data Products wrapped in a Level0 Context. If you want convert in one single step all the level 0 products contained in an ObservationContex, use this parameter to give as input the Level0Context contained in the ObservationContext. This is the primary input of the task.
level0block	Input Level-0 SPIRE Data Products wrapped in a Level0BlockContext. Use this parameter if you want to give as input level 0 products belonging to a single Building Block.
rawData	Input Level-0 SPIRE Data Products as a map. Use this parameter if you want to give as input level 0 products which are not ordered in building blocks.

3.2.3. Output Data Products

Level 0.5 products	**Level 0.5 products** The output products of this module are the Level 0.5 products.

3.2.4. Input Calibration Products

cal	Spire Calibration Context. You can use this parameter to give to the task all calibration products warped in a calibration context. If a calibration product is provided using the specific parameter (see below), the corresponding product contained in the calibration context is ignored.
resetHist	DPU reset history calibration product.
offsetHist	Detector offset history calibration product.
chanGain	Channel Gain calibration product.
bolPar	Bolometer parameters calibration product.
chanMask	Channel Mask calibration product.
chanNum	Channel Number Mapping calibration product.
chanNomRes	Channel Nominal Resistances calibration product.

3.2.5. How to use the *Engineering Conversion* Pipeline

3.2.6. Parameter Options

Users can set the following parameters of the module:

tempStorage	Flag that determines whether to use (or not to use) a temporary storage (using the "ProductSink") during the processing. If this flag is set to true, the task will save the processed products into the ProductSink. This allows the processing of large observations without the need for excessive memory usage. Note that the ProductSink must be initialised before executing the task, if this flag is set to true.

3.2.7. Examples

In the following example, the `EngConversionTask` is used to convert all level 0 products of an ObservationContext.

```
  IA>> obs=.... #obs is the ObservationContext
  IA>>
  IA>> level0_5=engConversion(level0=obs.level["level0"], cal=obs.calibration)
  IA>>
  IA>> obs.level["level0_5"]=level0_5  #here we put the result into the
ObservationContext
```

Since `level0` is the first parameter of the `EngConversionTask`, we can simplify the sintax in:

```
  IA>> level0_5=engConversion(obs.level["level0"], cal=obs.calibration)
```

The following example is similar to the previous but here we use a ProductSink.

```
tmppool=TemporalPool.createTmpPool("tmppool",TemporalPool.CloseMode.DELETE_ON_CLOSE)
    ProductSink.getInstance().productStorage=ProductStorage(tmppool)
```

In the following example, we convert the level 0 products of a single building block and we request the task to use a specific Channel Mask calibration product.

```
  IA>> obs=.... #obs is the ObservationContext
  IA>> obsid=obs.obsid
  IA>> bbid=0xA0300001L
  IA>> block=obs.level["level0"].get(obsid,bbid)
  IA>> chanMask=.....
  IA>> level0_5=engConversion(level0block=block, cal=obs.calibration,
chanMask=chanMask)
  IA>>
  IA>> obs.level["level0_5"]=level0_5  #here we put the result into the
ObservationContext
```

3.2.8. Error messages

TBW. TBW.

TBW. TBW.

3.3. The *Engineering Conversion* Pipeline Step-by-Step: Split Raw Data Module

3.3.1. Module Description

The *Split Raw Data* Module seperates the Raw SPIRE Timelines (RST) into the individual Level 0 data products. Earlier versions of HIPE (version 2), the data in level 0 are divided for each individual building block (e.g. a RCHKT, a RNHKT, a RPDT). From HIPE version 3, in level 0 there is a single product (RST, i.e. Raw Spire Timeline) for each individual building block. So the level 0 context in ObservationContexts produced by HIPE version 2 (or before) and HIPE version 3 are different. The engineering conversion (in HIPE version 3) will check the format of the input level 0. If the level 0 was produced with data split per type (as in HIPE version 2), it passes the Level 0 products through inchanged. If the level 0 was produced with a single product per Building Block (as in HIPE version 3), it uses the *SplitRawDataTask* to split this product and produce products in the old format, and continue the processing as before. Note: It is necessary to handle the case of level 0 context with the old format otherwise the software would not be able to process old data.

3.3.2. Input Data Products

Level 0 Raw SPIRE Timelines (RST)	Level 0 Raw SPIRE Timelines (RST) to be reformatted.

3.3.3. Output Data Products

Individual Level 0 style products	**Individual Level 0 style products** The single Level 0 Raw SPIRE Timelines (RST) are seperated into raw detector and raw non-detector (Housekeeping, auxiliary timelines).

3.3.4. Input Calibration Products

None

3.3.5. How to use the *Split Raw Data* Module

The `SplitRawDataTask` module

3.3.6. Parameter Options

Users can set the following parameters of the module:

3.3.7. Examples

In the following example, ...

```
IA>>
IA>>
IA>>
IA>>
IA>>
IA>>
```

3.3.8. Error messages

TBW. TBW.

TBW. TBW.

3.4. The *Engineering Conversion* Pipeline Step-by-Step: Converting Level 0 Products to Level 0.5 format

3.4.1. Module Description

The *Format Conversion* Module implements the reformat of Level 0 products into Level 0.5 products format.

3.4.2. Input Data Products

rawData	Input Level-0 SPIRE Data Product to be reformatted.

3.4.3. Output Data Products

Level 0.5 style products	**Level 0.5 style products** The data is reformatted into the style of Level 0.5 products.

3.4.4. Input Calibration Products

3.4.5. How to use the *Format Conversion* Module

The `FormatConversionTask`

3.4.6. Parameter Options

Users can set the following parameters of the module:

3.4.7. Examples

In the following example, we reformat a level 0 product into level 0.5 format.

```
  IA>> obs=.... #obs is the ObservationContext
  IA>> obsid=obs.obsid
  IA>> bbid=0xA0300001L
  IA>> rpdt=obs.level["level0"].get(obsid,bbid).rpdt # extract the RPDT of
building block 0xA0300001
  IA>>
  IA>> pdt=formatConversion(rawData=rpdt) # or you can do also
pdt=formatConversion(rpdt)
```

3.4.8. Error messages

TBW. TBW.

TBW. TBW.

3.5. The *Engineering Conversion* Pipeline Step-by-Step: Flagging out of range detectors

3.5.1. Module Description

The *Check ADC flags and truncation* Module implements the checking of ADC flags and truncation. An "ADC latch-up" happens when a cosmic ray hits the ADC (analogue to digital converter); the cosmic ray generates changes in the electronics corrupting the ADC reading. The electronics however detects these events and set the bit value in the ADCFLAGS corresponding to the affected ADC channel.

3.5.2. Input Data Products

rawData	Input detector timeline to be processed.

3.5.3. Output Data Products

Level 0.5 style products	Level 0.5 style products.

3.5.4. Input Calibration Products

3.5.5. How to use the *Check ADC flags and truncation* Module

The `checkAdcLatchTrunc` task

3.5.6. Parameter Options

Users can set the following parameters of the module:

3.5.7. Examples

In the following example, we process a PDT.

```
IA>> obs=.... #obs is the ObservationContext
IA>> obsid=obs.obsid
```

```
  IA>> bbid=0xA0300001L
  IA>> rpdt=obs.level["level0"].get(obsid,bbid).rpdt # extract the RPDT of
building block 0xA0300001
  IA>>
  IA>> pdt=formatConversion(rpdt) # reformatting
  IA>> pdt=checkAdcLatchTrunc(pdt)
```

3.5.8. Error messages

TBW. TBW.

TBW. TBW.

3.6. The *Engineering Conversion* Pipeline Step-by-Step: Masking Bad Detector Channels

3.6.1. Module Description

The *Mask Bad Channels* Module implements the masking of bad channels.

3.6.2. Input Data Products

data	Input detector timeline to be processed.

3.6.3. Output Data Products

Level 0.5 style products	**Level 0.5 style products** .

3.6.4. Input Calibration Products

chanMask	Channel Mask calibration product.
chanNum	Channel Number Mapping calibration product.

3.6.5. How to use the *Mask Bad Channels* Module

The `MaskBadChanTask` implements the masking of bad channels.

3.6.6. Parameter Options

Users can set the following parameters of the module:

3.6.7. Examples

In the following example, we process a PDT using only the channel mask. The PDT doesn't contain disconnected channels.

```
  IA>> obs=.... #obs is the ObservationContext
  IA>> obsid=obs.obsid
  IA>> bbid=0xA0300001L
  IA>> rpdt=obs.level["level0"].get(obsid,bbid).rpdt # extract the RPDT of
building block 0xA0300001
  IA>>
  IA>> pdt=formatConversion(rpdt) # reformatting
  IA>> pdt=detFlagger(pdt)
  IA>> pdt=maskBadChan(pdt,chanMask=obs.calibration.phot.chanMask)
```

In the following example, we process a SDT using both the channel mask and the channel number mapping. The latter will be used to remove disconnected channels.

```
  IA>> sdt=maskBadChan(sdt,chanMask=obs.calibration.spec.chanMask,
chanNum=obs.calibration.spec.chanNum)
```

Note that if you use a spectrometer calibration product on a photometer detector timeline (or viceversa) the task will throw a `SignatureException`

3.6.8. Error messages

TBW. TBW.

TBW. TBW.

3.7. The *Engineering Conversion* Pipeline Step-by-Step: Time Conversion

3.7.1. Module Description

The *Time Conversion* Module implements the time conversion and reordering.

3.7.2. Input Data Products

rawData	Input timeline to be processed.

3.7.3. Output Data Products

Level 0.5 style products	**Level 0.5 style products** .

3.7.4. Input Calibration Products

resetHist	DPU reset history calibration product.

3.7.5. How to use the `TimeConvReordTask`

The `TimeConvReordTask` implements the time conversion and reordering.

3.7.6. Parameter Options

Users can set the following parameters of the module:

3.7.7. Examples

In the following example, we process a PDT.

```
  IA>> obs=.... #obs is the ObservationContext
  IA>> obsid=obs.obsid
  IA>> bbid=0xA0300001L
  IA>> rpdt=obs.level["level0"].get(obsid,bbid).rpdt # extract the RPDT of
building block 0xA0300001
  IA>>
  IA>> pdt=formatConversion(rpdt) # reformatting
  IA>> pdt=detFlagger(pdt)
  IA>> pdt=maskBadChan(pdt,chanMask=obs.calibration.phot.chanMask)
  IA>> pdt=timeConvReord(pdt,resetHist=obs.calibration.resetHist)
```

In the following example, we process a Raw Nominal Housekeeping Timeline. Note that in this case we don't need a Reset History calibration product.

```
  IA>> obs=.... #obs is the ObservationContext
  IA>> obsid=obs.obsid
  IA>> bbid=0xA0300001L
  IA>> rpdt=obs.level["level0"].get(obsid,bbid).rnhkt # extract the RNHKT of
building block 0xA0300001
  IA>>
  IA>> nhkt=formatConversion(rnhkt) # reformatting
  IA>> nhkt=timeConvReord(nhkt)
```

3.7.8. Error messages

TBW. TBW.

TBW. TBW.

3.8. The *Engineering Conversion* Pipeline Step-by-Step: Calculating the JFET Voltages

3.8.1. Module Description

The *Calculate JFET Voltages* Module implements the conversion of detector signals from ADUs to JFET voltages.

3.8.2. Input Data Products

rawData	Input detector timeline to be processed.

nhkt	Nominal Housekeeping Timeline. If the NHKT is provided, the bias frequency is taken from this product and not from the metadata of the detector timeline (if it was set).

3.8.3. Output Data Products

Level 0.5 style products	**Level 0.5 style products** .

3.8.4. Input Calibration Products

offset	Detector offset history calibration product.
chanGain	Channel gains calibration product. This parameter is mandatory.

3.8.5. How to use the *Calculate JFET Voltages* Module

The `CalcJfetVoltTask` implements the conversion of detector signals from ADUs to JFET voltages.

3.8.6. Parameter Options

biasFreq	The bias frequency in Hz. If this value is set, the `nhkt` parameter is ignored as well as the bias frequency value in the input detector timeline metadata.

3.8.7. Examples

In the following example, we process a Photometer Detector Timeline, providing the offset history, the channel gain and the housekeeping. In this case, the voltage will be computed using equation (2) of the SPIRE Pipeline Description document. The bias frequency will be obtained from the Nominal Housekeeping Timeline.

```
IA>> pdt=... #pdt is a Photometer Detector Timeline
IA>> offset=... #the offset history
IA>> chanGain=... #the channel gain table
IA>> nhkt=... #  Nominal Housekeeping Timeline
IA>>
IA>> pdt=calcJfetVolt(pdt, offset=offset, chanGain=chanGain, nhkt=nhkt)
```

In the following example, we process a Photometer Detector Timeline, providing the offset history, the channel gain and a bias frequency. As before, the voltage will be computed using equation (2) of the SPIRE Pipeline Description document.

```
IA>> pdt=... #pdt is a Photometer Detector Timeline
IA>> offset=... #the offset history
IA>> chanGain=... #the channel gain table
IA>>
IA>> pdt=calcJfetVolt(pdt, offset=offset, chanGain=chanGain, biasFreq=131.7)
```

As the previous example, but setting the bias frequency in the input PDT.

```
IA>> pdt=... #pdt is a Photometer Detector Timeline
IA>> offset=... #the offset history
IA>> chanGain=... #the channel gain table
IA>> pdt.biasFreq=131.7
IA>> pdt=calcJfetVolt(pdt, offset=offset, chanGain=chanGain)
```

As the previous example, but without providing the offset history. In this case the voltage will be computed assuming an offset of 0 for all channels.

```
IA>> pdt=... #pdt is a Photometer Detector Timeline
IA>> chanGain=... #the channel gain table
IA>> pdt.biasFreq=131.7
IA>> pdt=calcJfetVolt(pdt, chanGain=chanGain)
```

3.8.8. Error messages

TBW. TBW.

TBW. TBW.

3.9. The *Engineering Conversion* Pipeline Step-by-Step: Calculating the bolometer RMS voltage and resistance

3.9.1. Module Description

The *Calculate Bolometer RMS Voltage* Module implements the calculation of bolometer RMS voltage and resistance from the JFET voltages.

3.9.2. Input Data Products

data	Input detector timeline to be processed.
nhkt	Nominal Housekeeping Timeline. The NHKT is needed to get the bias amplitudes. It is a mandatory input.

3.9.3. Output Data Products

Level 0.5 style products	**Level 0.5 style products** .

3.9.4. Input Calibration Products

chanNum	Channel number mapping calibration product.
chanGain	Channel gains calibration product. This parameter is mandatory.
balPar	Bolometer parameters calibration product.
chanNomRes	Channel nominal resistances calibration product.

3.9.5. How to use the *Calculate JFET Voltages* Module

The `CalcRmsVoltResTask` implements the calculation of RMS bolometers voltage and resistance from JFET voltages.

3.9.6. Parameter Options

Users can set the following parameters of the module:

3.9.7. Examples

In the following example, we process a Photometer Detector Timeline, providing only the channel gain and the housekeeping. In this case only the voltages can be computed; this will be done with the equation $V_{d\text{-}RMS}=V_{JFET\text{-}RMS}/H_{JFET}$.

```
IA>> pdt=... #pdt is a Photometer Detector Timeline
IA>> chanGain=... #the channel gain table
IA>> nhkt=... #  Nominal Housekeeping Timeline
IA>>
IA>> pdt=calcRmsVoltRes(pdt, chanGain=chanGain, nhkt=nhkt)
```

In the following example, we also provide the bolometer parameters table. Voltages and resistances will be computed using equations in section 3.8 of the SPIRE Pipeline Description document; but assuming a phase shift $\Delta\varphi=0$. Since the channel number mapping is not provided, the tasks assumes that thermal control channels are those with a name starting as "PTC" or "PMWP". For thermal control channels, the resistance is set to NaN, and the voltage is simply $V_{d\text{-}RMS}=V_{JFET\text{-}RMS}/H_{JFET}$.

```
IA>> pdt=... #pdt is a Photometer Detector Timeline
IA>> nhkt=... #  Nominal Housekeeping Timeline
IA>> chanGain=... #the channel gain table
IA>> bolPar=... #the bolometer parameters table
IA>>
IA>> pdt=calcRmsVoltRes(pdt, chanGain=chanGain, nhkt=nhkt, bolPar=bolPar)
```

In the following example, we also provide the channel nominal resistances and the channel number mapping product. Voltages and resistances and phase shifts will be computed using equations in section 3.8 of the SPIRE Pipeline Description document. Thermal control channels names will be taken form the channel number mapping product.

```
IA>> pdt=... #pdt is a Photometer Detector Timeline
IA>> nhkt=... #  Nominal Housekeeping Timeline
IA>> chanGain=... #the channel gain table
IA>> bolPar=... #the bolometer parameters table
IA>> chanNomRes=... #the channel nominal resistances table
IA>> chanNum=... #the channel number mapping table
IA>>
IA>> pdt=calcRmsVoltRes(pdt, chanGain=chanGain, nhkt=nhkt, bolPar=bolPar,
chanNum=chanNum, chanNomRes=chanNomRes)
```

3.9.8. Error messages

TBW. TBW.

TBW. TBW.

3.10. The *Engineering Conversion* Pipeline Step-by-Step: Adding Pointing Meta-Data Information to the Data

3.10.1. Module Description

The *Add Pointing Meta-Data* Module implements the computation of pointing metadata parameters.

3.10.2. Input Data Products

data	Input detector timeline to be processed.
nhkt	Nominal Housekeeping Timeline.

3.10.3. Output Data Products

Level 0.5 style products	**Level 0.5 style products** .

3.10.4. Input Calibration Products

No calibration data are needed.

3.10.5. How to use the *Add Pointing Meta-Data* Module

The *Add Pointing Meta-Data* Module implements the computation of pointing metadata parameters by the `AddPointingParamTask` task.

3.10.6. Input Control Parameters

step	The STEP value. If this value is set, the `nhkt` is ignored.

3.10.7. Examples

In the following example, we process a Photometer Detector Timeline, providing the housekeeping.

```
IA>> pdt=... #pdt is a Photometer Detector Timeline
IA>> nhkt=... #  Nominal Housekeeping Timeline
IA>>
IA>> pdt=addPointingParam(pdt, nhkt=nhkt)
```

In the following example, we provide directly the STEP value.

```
IA>> pdt=... #pdt is a Photometer Detector Timeline
IA>>
IA>> pdt=addPointingParam(pdt, step=12345)
```

3.10.8. Error messages

TBW. TBW.

TBW. TBW.

Chapter 4. Processing Data with the Photometer Large Scan Map Pipeline

4.1. Introduction

A detailed description of the photometer pipeline design can be found in

Griffin M., Dowell D., Lim T., Bendo G., Bock J., Cara C., Castro-Rodriguez N., Chanial P., Clements D., Gastaud R., Guest S., Glenn J., Hristov V., King K., Laurent G., Lu N., Mainetti G., Morris H., Nguyen H., Panuzzo P., Pearson C., Pinsard F., Pohlen M., Polehampton E., Rizzo D., Schulz B., Schwartz A., Sibthorpe B., Swinyard B., Xu K., Zhang L., The Herschel-SPIRE Photometer Data Processing Pipeline, Proc. SPIE, Space Telescopes and Instrumentation: Optical, Infrared, and Millimeter 7010, (2008)

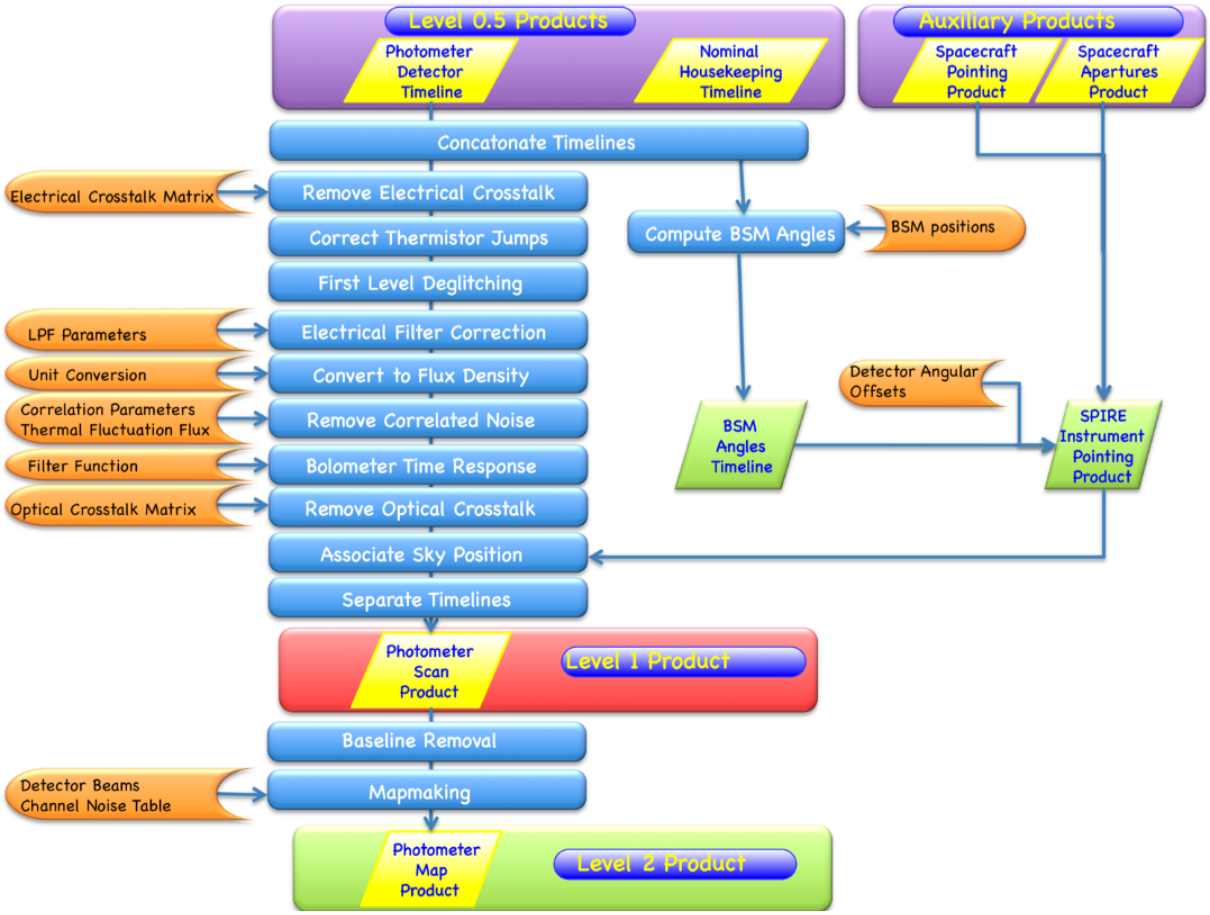

Figure 4.1. Flowchart for the Photometer Scan Map Pipeline.

4.2. Conversion of the BSM telemetry into Angles on the Sky

4.2.1. Module Description

The *Compute BSM Angles* Module converts the angles of the Beam Steering Mirror (BSM) to physical units (decimal degrees on the sky). The method is simple; interpolation from a calibration table containing the two converted angles (spacecraft Y and Z axes) versus the two raw signal values (in the BSM chop and jiggle axes). Note that for scan map observations there is no chopping, and therefore there is no need to create a "chop-jiggle timeline" as in the case of Jiggle observations. However, it is still necessary to create a BSM angle timeline because it is important to know where the BSM is pointing even if it supposed be at a fixed position. For scan map observations, the instrument does not produce BSM telemetry packets at all, but rather the input comes from the Nominal Housekeeping Timeline (NHK) which contains the BSM sensor signal values at a lower sampling rate.

4.2.2. Input Data Products

NHKT — **Nominal House Keeping Timeline** This contains the timeline of the BSM sensor signal values (with the chop axis aligned with the spacecraft Y-direction and the jiggle axis aligned with the Z-direction), in raw format in the `chopsenssig` and `jiggsenssig` columns.

4.2.3. Output Data Products

BAT — **BSM Angles Timeline** It contains timelines of angular distance on the sky from the BSM rest position in spacecraft Y and Z coordinates. For the definition see: http://www.spire.rl.ac.uk/icc/product_definitions/

4.2.4. Input Calibration Products

SCalPhotBsmPos — **BSM Position Table** This Table contains BSM sensor signal in raw units (in the chop and jiggle directions) and the angular distance on the sky from its rest position (in spacecraft Y and Z coordinates). There is one table for the Photometer and another for the Spectrometer, because the reference BSM position can be different. For the product definition see: http://www.spire.rl.ac.uk/icc/product_definitions/

4.2.5. How to use the *Compute BSM Angles* Module

The module takes as input the NHKT, and returns as output the BAT timeline, as descibed above.

4.2.6. Parameter Options

There are no parameter options.

4.2.7. Examples

```
bat = calcBsmAngles(nhkt, bsmPos=obs.calibration.phot.bsmPos)
```

The process is shown pictorially in the figure below. Note that the interpolation method is not a parameter for the moment.

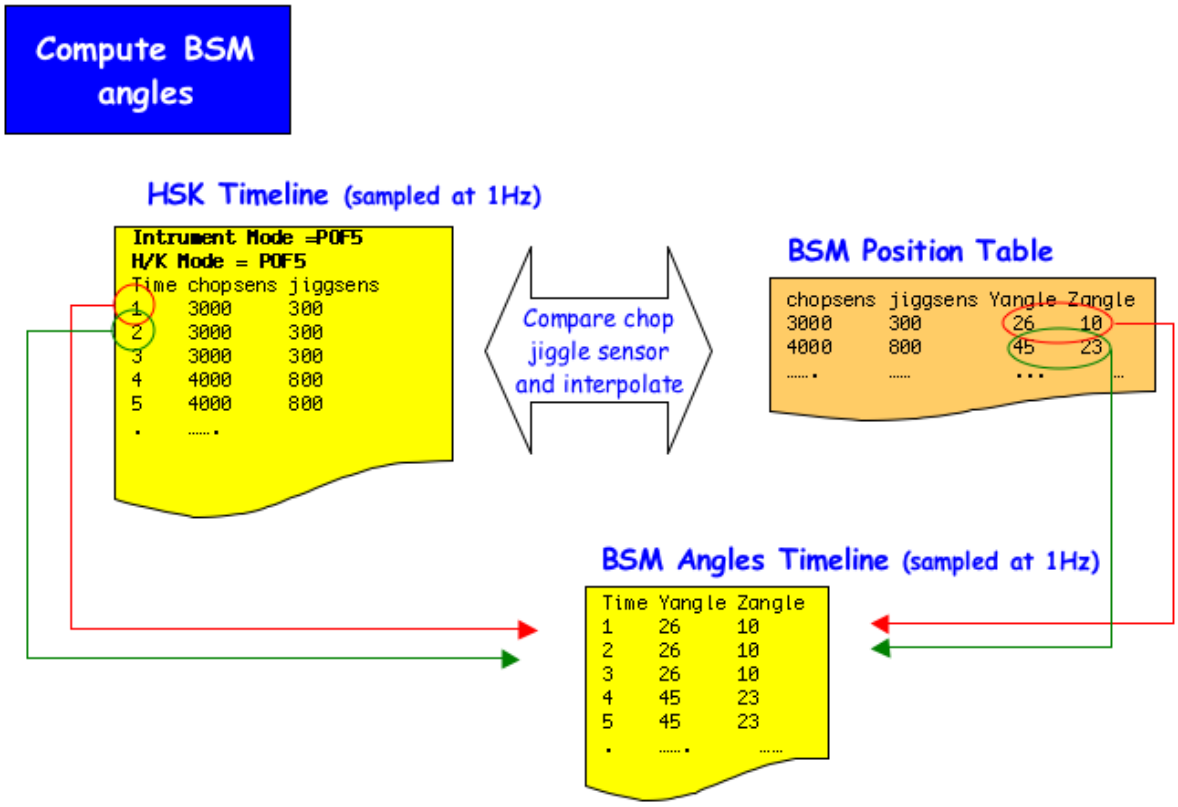

Figure 4.2. Creation of BSM Angles Timeline by the Compute BSM Angles module in the Scan Map Pipeline.

4.2.8. Error messages

error: herschel.ia.task.SignatureException: Error in calcBsmAngles Task:the input product must be either a Beam Steering Mirror Timeline (BSMT) or a Nominal Housekeeping Timeline (NHKT). The input product type was checked and found not to be a BSM; timeline or a NHKT;. The module can only operate on these two product types.

4.3. SPIRE Pointing Product

4.3.1. Module Description

This class implements the Spire Pointing Product.

4.3.2. Input Observational Data Products

The Spire Pointing Product is a context containing the Herschel Pointing Product, the SIAM product, the DetAngOff product and the Bsm Angle Timeline. This class allows the user to obtain the position of one or more SPIRE detectors on the sky at a given time.

No product definition document available yet.

4.3.3. Output Data Products

The module outputs a Spire Pointing Product.

The SpirePointingProduct calculates the pointing for each detector by adding the value stored in the DetAngOff table (gives the offset of each detector on the sky from the instrument boresight) and adding the value from the BSM Angles Timeline to the pointing of the central aperture, which is derived from the BSM Positions calibration from the measured CHOP and JIGG sensor values.

The BSM Positions calibration is calculated by observing what CHOPSENS and JIGGSENS place the source (i.e. the instrument boresight, or HPP+SIAM) on a particular detector, it gives the angle on the sky through which the instrument boresight has moved on the sky. In order to calculate the final pointing at each detector in the array, the different contributions to the pointing would need to be combined as follows:

HPP + SIAM + DetAngOff - BAT

where the BAT is subtracted so that it gives the angle that the boresight has moved on the sky, which is in the opposite sense to the correction that must be applied to the actual sky coordinates for each detector.

See the Herschel Products Definition Document or alternatively http://www.spire.rl.ac.uk/icc/product_definitions/.

4.3.4. Input Calibration Products Required

No calibration products required.

4.3.5. Parameter options

Users can set the following parameters of the Pointing Product module:

4.3.6. Examples

Creation:

```
spp =
createSpirePointingProduct(hpp=obs.photsiliary.pointing,siam=obs.photsiliary.siam,
detAngOff=obs.calibration.phot.detAngOff, bat=bat)
```

Obtain the Gyro propagated sky position of the PSWE8 and PSWE4 at a given time

```
pdt=obs.level0_5.getProduct(5).pdt
ra,dec=spp.getSkyPosition(["PSWE8","PSWE4"],pdt.sampleTime[0],"gyro")
```

4.3.7. Error messages

4.4. Deglitching the Timeline Data

4.4.1. Module Description

The *First Level Deglitcing* Module removes glitches due to cosmic ray hits or other impulse-like events. The basic assumption is that the glitch signature is similar to a Dirac delta function. This

process is composed of two steps: the first step implements a wavelet-based local regularity analysis to detect glitch signatures in the measured signal; the second step locally reconstructs a signal free of such glitch signatures.

The wavelet method for deglitching is described in detail in;

1. Ordenovic C., Surace C., Torresani B., Llebaria A., Detection of glitches and signal reconstruction using Hoelder and wavelet analysis, Statistical Methodology, 2008, Volume 5, Issue 4, 373-386

2. Ordenovic C., Surace C., Torresani B., Llebaria A., Baluteau J.P., Use of a local regularity analysis by a wavelet analysis for glitch detection, SPIE, 2005, 5909, 556-567

4.4.2. Input Data Products

PDT; **Photometer Detector Timeline** The input PDT product contains timelines for each photometer detector for each observation building block.

4.4.3. Output Data Products

PDT; **Spectrometer Detector Timeline** The output PDT product contains deglitched timelines for each spectrometer detector for each observation building block.

4.4.4. Input Calibration Products

No calibration data are needed.

4.4.5. How to use the *First Level Deglitching* Module

4.4.6. Parameter Options

The first level deglitching task employs a complex algorithm, based on a continuous wavelet transform with a Mexican Hat wavelet and subsequent complex processing for a local regularity analysis. A thorough understanding of this algorithm is required in order to make good choices when setting the involved parameters. Users are strongly discouraged from changing the default value of these parameters unless they have closely studied the extended documentation for this task.

scaleMin, scaleMax, and voices — scaleMin/scaleMax are the minimum/maximum values of the range of wavelet scales used for the linear fit to the Wavelet Transform Modulus Maxima Lines. scaleMin and scaleMax should both be positive integers and scaleMin should be less than scaleMaxe. The current best estimation for the Scanmap Pipeline are values of scaleMin = 1, scaleMax= 8 and voices = 5.. voices defines how many wavelet scales are calculated for a range of one within the scale range. The size of the range defined by scaleMin and scaleMax times the value of scaleInterval is directly proportional to execution time. Both should therefore be kept at the minimum required to accurately determine the Holder index.

holderMax and holderMin — holderMax/hmin are the minimum/maximum values of a real data type of the range of slopes which are interpreted as glitch indicators. The range should include -1 and holderMax should be larger than hmin. The current best estimation for the Scanmap Pipeline are values of holderMax = -0.3 and holderMin = -1.9. The range defined by holderMax and hmin allows to select between a more sensitive glitch detection to detect even weak glitches, or a more conservative

glitch detection to avoid false positives, i.e. samples that are flagged as glitches which are arguably due to other causes.

correlationThreshold — This parameter relates to the square of the correlation coefficient of the linear fit to the Wavelet Transform Modulus Maxima Line. It sets a lower threshold of the correlation to enforce a minimum goodness of the fit to the data. This real number should be between 0 and 1, and is usually selected closer to 1. A value of 0 removes any selection criterion within the algorithm for goodness of fit. A value of 1 will make the criterion so stringent that only a perfect fit will be accepted. The current best estimation for the Scanmap Pipeline are values of correlationThreshold = 0.7. The value selected for correlationThreshold allows to select between a more sensitive glitch detection to detect even weak glitches, or a more conservative glitch detection to avoid false positives, i.e. samples that are flagged as glitches which are arguably due to other causes.

4.4.7. Examples

```
IA>> from herschel.spire.ia.pipeline.common.deglitch import DeglitchingTask
IA>>
IA>> deglitch=DeglitchingTask()
IA>> pdt=deglitch(pdt,
                      scaleMin=1,
     scaleMax=8,
     voices=5,
     holderMin=-1.9,
     holderMax=-0.3,
     correlationThreshold=0.7)
IA>>
```

4.4.8. Error messages

severe :unable to store output product. The task failed to write the output product.

warning :Unable to perform reconstruction. The task failed to reconstruct signal samples.

severe :unable to perform wavelet decomposition.. The task failed to identify glitches because the signal could not be decomposed into its wavelet components.

4.5. Removing Electrical Crosstalk from the Timeline Data

4.5.1. Module Description

The *Electrical Cross Talk Correction* module removes electrical crosstalk from the Detector Timelines. The module multiplies a crosstalk removal matrix with the signal vector for each time sample and returns Detector Timelines which are corrected for electrical crosstalk.

4.5.2. Input Data Products

PDT — **Photometer Detector Timeline** The input PDT product contains timelines for each detector for each observation building block.

4.5.3. Output Data Products

PDT — **Photometer Detector Timeline** The output PDT product contains timelines for each detector for each observation building block with the signal corrected for electrical crosstalk.

4.5.4. Input Calibration Products

SCalElecCross — **Electrical Crosstalk Matrix** There are three electrical crosstalk matrices one for each of the SPIRE photometer arays. They contain entries for the electrical crosstalk between the detectors of each one of the two detector arrays. In the absence of any crosstalk, the matrices contain unity elements in the diagonals and zero elements anywhere else.

4.5.5. How to use the *Remove Electrical Crosstalk* Module

It should be noted that, prior to launch, no electrical crosstalk was observed. During the fabrication of the warm read-out electronics, spot-checking showed extremely low levels of electrical crosstalk. An additional analysis was performed for the detector arrays of the SPIRE imaging photometer using data from the ground-based test campaigns. This analysis checked whether the random instances of detectors absorbing ionizing radiation lead to reproducible signals in other detectors. Again, this analysis has shown no measurable evidence for electrical crosstalk. Close examination of the post-launch performance will be required to decide whether the detector arrays suffer significant electrical crosstalk. Until then, this module simply multiplies the detector timelines by the identity matrix.

4.5.6. Parameter Options

N/A. There are no options for this module.

4.5.7. Examples

Assuming that `pdt` is an PDT; product and `obs` an observation context:

```
pdt = elecCrossCorrection(pdt, elecCross=obs.calibration.phot.elecCross)
```

4.5.8. Error messages

TBW. TBW.

TBW. TBW.

4.6. Correcting for Electrical Filter Response

4.6.1. Module Description

The *Correction for Electrical Filter Response* module is responsible for removing the effects of the electrical filter (Low pass filter) transient response. This is achieved by:

1. Transforming the signal timelines of individual detectors in to the Fourier domain
2. Dividing by the electrical filter transfer function (described in AD05, section 3.5.3)
3. Transforming the result back to the time domain

4.6.2. Input Data Products

PDT	**Photometer Detector Timeline** The input PDT product contains timelines for each detector for each observation building block.

4.6.3. Output Data Products

PDT	**Photometer Detector Timeline** The output PDT product contains timelines for each detector for each observation building block with the signal corrected for electrical filter response.

4.6.4. Input Calibration Products

SCalPhotElecFiltCorr	**Electrical Filter Correction** The calibration product provides information on the parameters describing the bolometer transfer function (AD01, section 4.1.5).

4.6.5. How to use the *Correction for Electrical Filter Response* Module

4.6.6. Parameter Options

N/A. There are no options for this module.

4.6.7. Examples

```
                                IA>> from
herschel.spire.ia.pipeline.phot.elecfiltcorr import lpfResponseCorrection
                                IA>>
                                IA>>
Mytask=lpfResponseCorrection.lpfResponseCorrection()
                                IA>>
                                IA>> PDT_IN=Mytask(PDT_IN) # where PDT_IN is the
input Phot. Det. Timeline product
                                IA>> #The above command will correct the timelines
included in the input product
                                IA>> #for the effects of the electrical filter.
```

4.6.8. Error messages

TBW. TBW.

TBW. TBW.

4.7. Converting the Detector Timelines into Flux Units

4.7.1. Module Description

The *Flux Conversion* Module applies a correction for the nonlinear bolometer responsivity to a photometer detector voltage time line V (i.e., PDT) by integrating the function, f(V) = K1 + K2/(V-K3), from Vo to V, where the parameters Vo, K1, K2, and K3 are given for each and every detector channel in a calibration product (`SCalPhotFluxConv()`. The result is an in-beam flux density timeline.

4.7.2. Input Data Products

PDT	**Photometer Detector Timeline** Photometer Detector Timeline (PDT) with bolometer signal in electrical units (i.e. Volts)

4.7.3. Output Data Products

PDT	**Photometer Detector Timeline** Photometer Detector Timeline (PDT) product with bolometer signal in units of Jy

4.7.4. Input Calibration Products

ScalPhotFLuxConv	**Photometer Flux Conversion and Non-linearity Correction Coefficients** contains, for each bolometer channel, V0, K1, K2, K3 and their uncertainties, as well as Vmin and Vmax, the calibrated limiting bolometer voltages.

4.7.5. How to use the *Flux Conversion* Module

4.7.6. Parameter Options

There are no parameter options for this module.

4.7.7. Examples

Jython Usage Example: Assuming a detector timeline of SPIRE photometer data processed to Level 0.5, with bolometer signal data in units of volts `PDT` and a calibration product, `fluxConv`.

```
outputfluxproduct = photFluxConversion(pdt,fluxConv=obs.calibration.phot.fluxConv)
```

4.7.8. Error messages

Error messages generated:

If the calculations result in taking the log of a negative number, the exception will be noted and the point generating the exception will be flagged in the output product mask.

4.8. Removing Correlated Noise from the Detector Timelines

4.8.1. Module Description

The *Remove Correlated Noise* Module subtracts low frequency (< 1Hz) noise, caused by variations of the detector array bath temperature, from scan mapping data time lines.The module is based on the empirical correlations between detectors and thermistors (or, in case of high bias voltages when thermistors are saturated, the correlations between detectors and dark pixels). The module shall be run after the correction for any nonlinearity (the flux conversion module).

4.8.2. Input Data Products

PDT — **Photometer Detector Timeline** Photometer Detector Timeline (PDT) with bolometer signal converted to flux units (e.g. Jy.) by Photometer Flux Conversion Module.

4.8.3. Output Data Products

PDT — **Photometer Detector Timeline** Data timelines in the units of Jy, corrected for temperature drift.

4.8.4. Input Calibration Products

ScalPhotTempDriftCorr — **Temperature Drift Correction Parameter Table** A product containing for each detector array, a thermistor selection flag, the base voltage values for each thermistor, the errors for those values, and a validity calibration flag for those values. It contains for each bolometer channel, correction coefficients A1, A1 error, B1, B1 error, AB1 flag (all for thermistor 1) and similar values for thermistor 2.

4.8.5. How to use the *Remove Correlated Noise* Module

4.8.6. Parameter Options

Users can set the following parameters of the module:

timeSpan — the timeSpan (in sec) used for averaging the thermistor timeline prior to performing spline function smoothing. Its default value is 5 (set within the task itself).

4.8.7. Examples

Jython Usage Example: Assuming a detector timeline of SPIRE photometer data, `PDT` processed by the Photometer Flux Conversion Module with bolometer signal data in units of Janskys. A tempdrift calibration product, `tempDrift`.

```
  pdt=temperatureDriftCorrection(pdt,
tempDriftCorr=obs.calibration.phot.tempDriftCorr)
```

4.8.8. Error messages

TBW. None

TBW. None

4.9. Correcting for the Bolometer Time Response

4.9.1. Module Description

The *Bolometer Time Response Correction* Module is responsible for removing the effects of the bolometer transient response. This is achieved by:

1. Transforming the signal timelines of individual detectors in to the Fourier domain
2. Dividing by the bolometer transient response transfer function (described in AD05, section 4)
3. Transforming the result back to the time domain

4.9.2. Input Data Products

PDT	**Photometer Detector Timeline** The input PDT product contains timelines for each detector for each observation building block.

4.9.3. Output Data Products

PDT	**Photometer Detector Timeline** The output PDT product contains timelines for each detector for each observation building block with the signal corrected for bolometer time response.

4.9.4. Input Calibration Products

ScalPhotDetTimeConst	**Photometer Detector Time Constants** Information on the parameters describing the bolometer transfer function will be provided by the(ScalPhotDetTimeConst) calibration product (AD01, section 4.1.8).

4.9.5. How to use the *Bolometer Time Response Correction* Module

4.9.6. Parameter Options

Users can set the following parameters of the module:

4.9.7. Examples

```
                                IA>> from
herschel.spire.ia.pipeline.phot.bolorespcorr import CorrBolTimeResponseTask
```

```
                                IA>>
                                IA>>
Mytask=CorrBolTimeResponseTask.CorrBolTimeResponseTask()
                                IA>>
                                IA>> PDT_IN=Mytask(PDT_IN) # where PDT_IN is the
input Phot. Det. Timeline product
                                IA>> #The above command will correct the timelines
included in the input product
                                IA>> #for the effects of the bolometer transient
response.
```

4.9.8. Error messages

TBW. TBW.

TBW. TBW.

4.10. Removing the Effects of Optical Crosstalk from the Data

4.10.1. Module Description

The *Optical Crosstalk Removal* Module

4.10.2. Input Data Products

Input Product name here **Input Product name here** and description to follow here

4.10.3. Output Data Products

Output Product Here **Output Product Name Here** and description to follow here

4.10.4. Input Calibration Products

Calibration Product Name Here **Calibration Product Name Here** and description to follow here

4.10.5. How to use the *Optical Crosstalk Removal* Module

4.10.6. Parameter Options

Users can set the following parameters of the module:

4.10.7. Examples

```
pdt = photOptCrossCorrection(pdt, optCross=obs.calibration.phot.optCross)
```

4.10.8. Error messages

4.11. Adding Positional Information (WCS) to the Data

4.11.1. Module Description

The *Associate Sky Position* Module

4.11.2. Input Data Products

Input Product name here **Input Product name here** and description to follow here

4.11.3. Output Data Products

Output Product Here **Output Product Name Here** and description to follow here

4.11.4. Input Calibration Products

Calibration Product Name Here **Calibration Product Name Here** and description to follow here

4.11.5. How to use the *Associate Sky Position* Module

4.11.6. Parameter Options

Users can set the following parameters of the module:

4.11.7. Examples

4.11.8. Error messages

TBW. TBW.

TBW. TBW.

4.12. Converting the On-Board Time to International Time Standards

4.12.1. Module Description

The *Time Correction* Module

4.12.2. Input Data Products

Input Product name here **Input Product name here** and description to follow here

4.12.3. Output Data Products

Output Product Here **Output Product Name Here** and description to follow here

4.12.4. Input Calibration Products

Calibration Product Name Here **Calibration Product Name Here** and description to follow here

4.12.5. How to use the *Time Correction* Module

4.12.6. Parameter Options

Users can set the following parameters of the module:

4.12.7. Examples

4.12.8. Error messages

TBW. TBW.

TBW. TBW.

4.13. Creating Image Scan-Maps

4.13.1. Module Description

The *Map Making* provides the map-making routines used by the SPIRE photometer pipeline. Scan mode timelines are combined to create a map using a choice of two algorithms: direct coaddition (`NaiveScanMapperTask`) and maximum-likelihood estimate (`MadScanMapperTask`). All output maps are North-oriented.

4.13.2. Input Observational Data Products

PSP — **Photometer Scan Product** The `NaiveScanMapperTask` and `MadScanMapperTask` take one Photometer Scan Product (PSP) or a context of PSP as input. For a definition, see:

http://www.spire.rl.ac.uk/icc/product_definitions/

4.13.3. Output Data Products

All map-making tasks returns the map associated to the bolometer array specified by the user. The map is stored as a `SimpleImage`. For a definition, see:

ftp://ftp.rssd.esa.int/pub/HERSCHEL/csdt/releases/doc/api/herschel/ia/dataset/image/SimpleImage.html

4.13.4. Input Calibration Products Required

The `MadScanMapperTask` requires a Dtector Noise Spectrum for thephotometer channel noise. For a definition, see:

http://www.spire.rl.ac.uk/icc/product_definitions/

For the `NaiveApppMapperTask` there are no calibration files required for the Jiggle Map `NaiveApppMapperTask`
.

4.13.5. How to use the *Map Making* Module

The module contains 3 mappers: 2 for the scan mode (`NaiveScanMapperTask` and `MadScanMapperTask`) and 1 for the jiggle mode (`NaiveApppMapperTask`).

The input parameters are a context of Photometer Detector Timelines (`PointedProduct`) for the scan mode and an Averaged PointedPhotometer Product for the jiggle mode.

The user should set the `array` keyword to "PSW", "PMW" or "PLW" to select the bolometer array to be processed. The optional keyword `maxmemory` can be used to increase the memory to be allocated for map-making and avoid slowing down by limiting I/O. The termination criteria for the task `MadScanMapperTask` can also be modified: the keywords `maxiterationpcg` and `maxrelerrorpcg` specify the maximum number of iterations and the maximum allowable residual in the Preconditionned Conjugate Gradient method.

The mappers only compute one map at a time, for the bolometer array specifed by the user. The output map is stored as a `SimpleImage` product, for which convenient visualisation tools are available (see examples below) within HCSS.

The mappers use temporary files to store the image coordinates (X,Y). `IOException` will be thrown is these temporary files can not be written or read. Out of memory exception can occur if the size of the map is too large (because of limited resource or incorrect astrometry)

4.13.6. Parameter options

Users can set the following parameters of the Map Making module:

array — **Name of the bolometer array to be processed** The value of this parameter can be "PSW", "PMW" or "PLW". If not set, the default bolometer array is assumed to be "PSW".

resolution	**Pixel size of the output map** The value of this parameter is in arcseconds. If not set, the default values are 6", 10" and 14 for the PSW, PMW and PLW bolometer array.
maxmemory	**Maximum memory in bytes to be used to hold the pointing matrix and the timeline in memory.** If the allocated memory isn't large enough, processing time may be I/O limited. By default, 1GB is assumed.
maxiterationpcg	**The maximum number of Preconditionned Conjugate Gradient (PCG) iterations before termination** (`MadScanMapperTask` only). By default, the maximum number of iterations is 50.
maxrelerrorpcg	**The largest allowable relative residual in PCG** (`MadScanMapperTask` only). By default, its value is 1e-6.

4.13.7. Examples

For scan mode observations, the user has to provide a set of building blocks, organised as a `Context` of level-1 `PointedProduct`.

Here is an example in which the PointedProduct Context is obtained from the Observation Context

```
IA>> from herschel.spire.ia.pipeline.phot.scanmap import NaiveScanMapperTask
IA>> mapper = NaiveScanMapperTask()
IA>> scanCon=obs.level1.getScan()
IA>> psw1=mapper(scanCon, array='PSW')
IA>> Display(psw1)
```

To use the maximum-likelihood mapper (MADMap), the synthax is similar.

```
IA>> from herschel.spire.ia.pipeline.phot.scanmap import MadScanMapperTask
IA>> mapper = MadScanMapperTask()
IA>> psw2 = mapper(scanCon, array='PSW')
IA>> Display(psw2)
```

4.13.8. Error messages

4.14. Point Source Extraction

4.14.1. Module Description

The *Point Source Extraction* Module

4.14.2. Input Observational Data Products

4.14.3. Output Data Products

4.14.4. Input Calibration Products Required

4.14.5. How to use the *Point Source Extraction* Module

4.14.6. Parameter options

Users can set the following parameters of the Demodulation module:

4.14.7. Examples

4.14.8. Error messages

Chapter 5. Processing Data with the Photometer Jiggle Map Pipeline

5.1. Introduction

A detailed description of the photometer pipeline design can be found in

Griffin M., Dowell D., Lim T., Bendo G., Bock J., Cara C., Castro-Rodriguez N., Chanial P., Clements D., Gastaud R., Guest S., Glenn J., Hristov V., King K., Laurent G., Lu N., Mainetti G., Morris H., Nguyen H., Panuzzo P., Pearson C., Pinsard F., Pohlen M., Polehampton E., Rizzo D., Schulz B., Schwartz A., Sibthorpe B., Swinyard B., Xu K., Zhang L., The Herschel-SPIRE Photometer Data Processing Pipeline, Proc. SPIE, Space Telescopes and Instrumentation: Optical, Infrared, and Millimeter 7010, (2008)

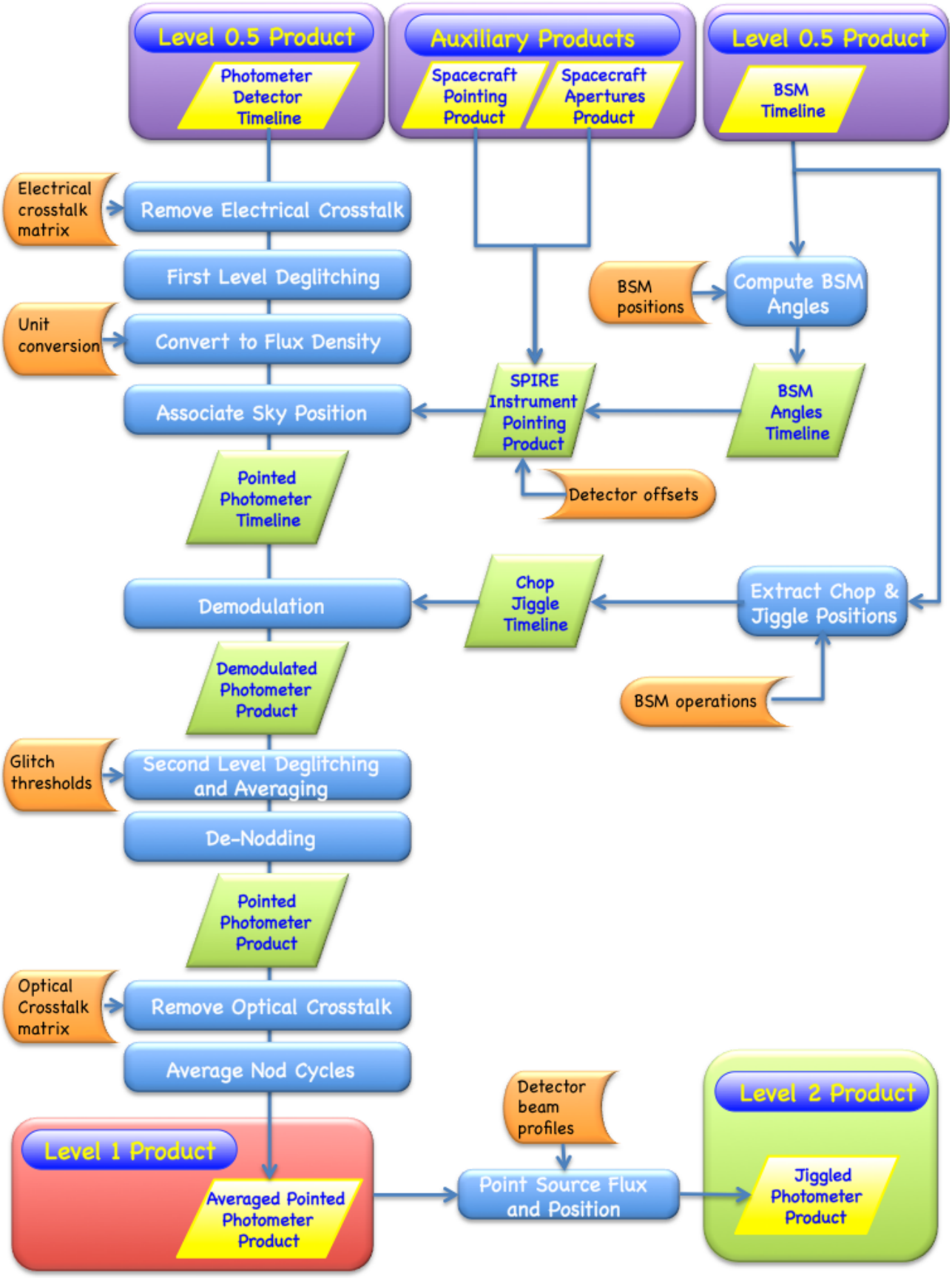

Figure 5.1. Flowchart for the Photometer Jiggle Map Pipeline.

5.2. Conversion of the BSM telemetry into Angles on the Sky

5.2.1. Module Description

The *Compute BSM Angles* Module converts the angles of the Beam Steering Mirror (BSM) to physical units (decimal degrees on the sky). The method is simple; interpolation from a calibration table containing the two converted angles (spacecraft Y and Z axes) versus the two raw signal values (in the BSM chop and jiggle axes).

5.2.2. Input Data Products

NHKT — **Nominal House Keeping Timeline** This contains the timeline of the BSM sensor signal values (with the chop axis aligned with the spacecraft Y-direction and the jiggle axis aligned with the Z-direction), in raw format in the `chopsenssig` and `jiggsenssig` columns.

5.2.3. Output Data Products

BAT — **BSM Angles Timeline** It contains timelines of angular distance on the sky from the BSM rest position in spacecraft Y and Z coordinates. For the definition see: http://www.spire.rl.ac.uk/icc/product_definitions/

5.2.4. Input Calibration Products

SCalPhotBsmPos — **BSM Position Table** This Table contains BSM sensor signal in raw units (in the chop and jiggle directions) and the angular distance on the sky from its rest position (in spacecraft Y and Z coordinates). There is one table for the Photometer and another for the Spectrometer, because the reference BSM position can be different. For the product definition see: http://www.spire.rl.ac.uk/icc/product_definitions/

5.2.5. How to use the *Compute BSM Angles* Module

The module takes as input the NHKT, and returns as output the BAT timeline, as descibed above.

5.2.6. Parameter Options

There are no parameter options.

5.2.7. Examples

```
bat = calcBsmAngles(nhkt, bsmPos=obs.calibration.phot.bsmPos)
```

The process is shown pictorially in the figure below. Note that the interpolation method is not a parameter for the moment.

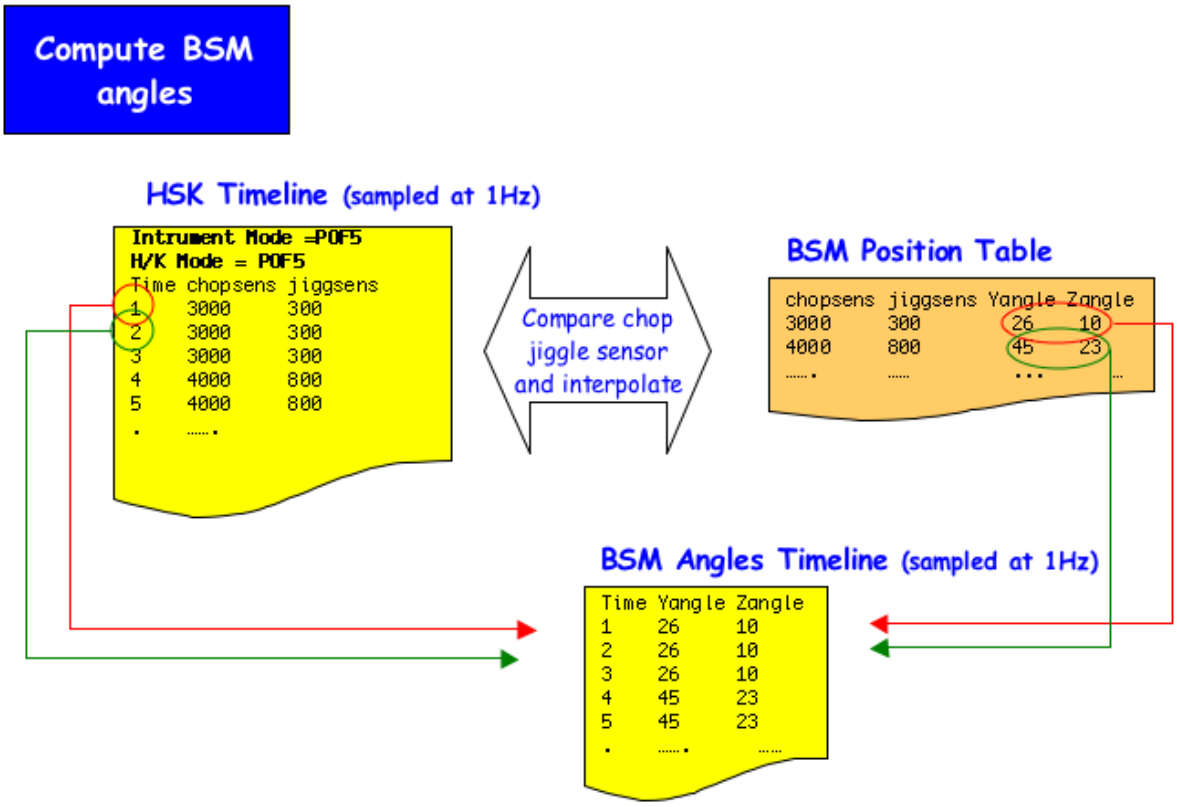

Figure 5.2. Creation of BSM Angles Timeline by the Compute BSM Angles module in the Scan Map Pipeline.

5.2.8. Error messages

error: herschel.ia.task.SignatureException: Error in calcBsmAngles Task:the input product must be either a Beam Steering Mirror Timeline (BSMT) or a Nominal Housekeeping Timeline (NHKT). The input product type was checked and found not to be a BSM; timeline or a NHKT;. The module can only operate on these two product types.

5.3. SPIRE Pointing Product

5.3.1. Module Description

This class implements the Spire Pointing Product.

5.3.2. Input Observational Data Products

The Spire Pointing Product is a context containing the Herschel Pointing Product, the SIAM product, the DetAngOff product and the Bsm Angle Timeline. This class allows the user to obtain the position of one or more SPIRE detectors on the sky at a given time. The SpirePointingProduct calculates the pointing for each detector by adding the value stored in the DetAngOff table (gives the offset of each detector on the sky from the instrument boresight) and adding the value from the BSM Angles Timeline to the pointing of the central aperture, which is derived from the BSM Positions calibration from the measured CHOP and JIGG sensor values. The BSM Positions calibration is calculated by observing

what CHOPSENS and JIGGSENS place the source (i.e. the instrument boresight, or HPP+SIAM) on a particular detector, it gives the angle on the sky through which the instrument boresight has moved on the sky. In order to calculate the final pointing at each detector in the array, the different contributions to the pointing would need to be combined as follows: HPP + SIAM + DetAngOff - BAT where the BAT is subtracted so that it gives the angle that the boresight has moved on the sky, which is in the opposite sense to the correction that must be applied to the actual sky coordinates for each detector.

No product definition document available yet.

5.3.3. Output Data Products

The module outputs a Spire Pointing Product.

The SpirePointingProduct calculates the pointing for each detector by adding the value stored in the DetAngOff table (gives the offset of each detector on the sky from the instrument boresight) and adding the value from the BSM Angles Timeline to the pointing of the central aperture, which is derived from the BSM Positions calibration from the measured CHOP and JIGG sensor values.

The BSM Positions calibration is calculated by observing what CHOPSENS and JIGGSENS place the source (i.e. the instrument boresight, or HPP+SIAM) on a particular detector, it gives the angle on the sky through which the instrument boresight has moved on the sky. In order to calculate the final pointing at each detector in the array, the different contributions to the pointing would need to be combined as follows:

HPP + SIAM + DetAngOff - BAT

where the BAT is subtracted so that it gives the angle that the boresight has moved on the sky, which is in the opposite sense to the correction that must be applied to the actual sky coordinates for each detector.

See the Herschel Products Definition Document or alternatively http://www.spire.rl.ac.uk/icc/product_definitions/.

5.3.4. Input Calibration Products Required

No calibration products required.

5.3.5. Parameter options

Users can set the following parameters of the Pointing Product module:

5.3.6. Examples

Creation:

```
spp =
createSpirePointingProduct(hpp=obs.photjiliary.pointing,siam=obs.photjiliary.siam,
detAngOff=obs.calibration.phot.detAngOff, bat=bat)
```

Obtain the Gyro propagated sky position of the PSWE8 and PSWE4 at a given time

```
pdt=obs.level0_5.getProduct(5).pdt
ra,dec=spp.getSkyPosition(["PSWE8","PSWE4"],pdt.sampleTime[0],"gyro")
```

5.3.7. Error messages

5.4. Extracting the Chop and Jiggle Positions from the BSM Timeline

5.4.1. Module Description

The *Extract Chop and Jiggle Positions* Module extracts from the raw values of the angles of the Beam Steering Mirror the position of the jiggle map and the chopper information. The BSM has a definite position, then move to another one. We have a start time and an end time for each position. The method is simple: each couple (chopper angle, jiggle angle) is compared to each position of the Operation calibration table with a tolerance. If it falls inside the rectangle defined by the tolerance, it is in this position, see figure below

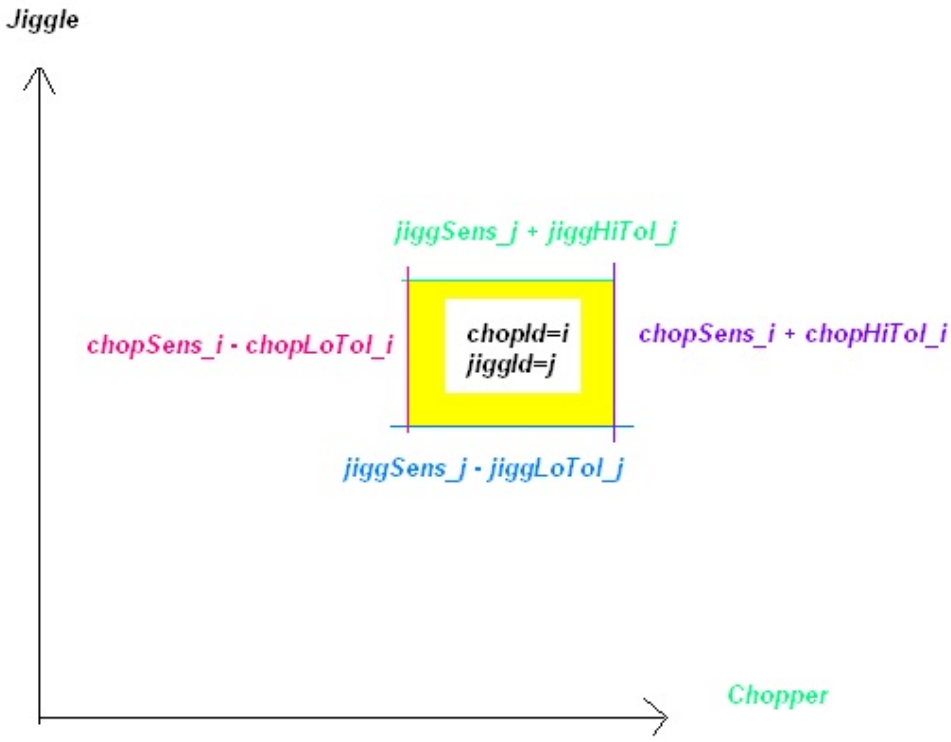

Figure 5.3. BSM Operation Table

5.4.2. Input Data Products

BSMT	**Beam Steering Mirror Timeline** This contains the timeline of the positions of the two mirrors, the chopper and the jiggle mirror, in raw

format. For the product definition see: http://www.spire.rl.ac.uk/icc/product_definitions/

5.4.3. Output Data Products

CJT — **Chop Jiggle Timeline** The final output of this module is Chopper Jiggle Timeline (CJT). It contains the start time and end time and the id of each consecutive jiggle map position. In addition, for debugging purpose, the converted values Y,Z of each jiggle map position is also stored. For the definition see: http://www.spire.rl.ac.uk/icc/product_definitions/

5.4.4. Input Calibration Products

SCalPhotBsmOps — **BSM Operations Table** This Table contains BSM sensor signal in raw units (in chop and jiggle directions) for each point of each jiggle map (remark: a 7 points jiggle map has 14 points because of the two chopper positions, On/Off). There is one table for the photometer and another for the spectrometer, because the reference bsm position can be different. For the product definition see: >http://www.spire.rl.ac.uk/icc/product_definitions/

5.4.5. How to use the *Extract Chop and Jiggle Positions* Module

The module takes a `EdpProduct` object as input. This contains the timeline of the two angles of the mirror (chop and jiggle) in raw data (BSMT).

The module needs two calibration products `BsmPos` and `BsmOps`.The first contains information for converting the raw angles, and the second the positions of the jiggle map.

The task output is a `Chop Jiggle Timeline` which contains the start time, end time, and id of each position in the jiggle map (CJT).

5.4.6. Parameter Options

Users can set the following parameters of the module:

5.4.7. Examples

Assuming that `bsmt` is a BSMT product, `bsmOps` is BsmOps product, and `bsmPos` is a BsmPos product. products:

```
calcBsmFlags = calcBsmFlags()
cjt = calcBsmFlags(bsmt=bsmt, bsmOps=bsmOps, bsmPos=bsmPos)
```

Note that CJT contains converted angles for debugging. This should vanish, and so the BsmPos calibration product should not be needed anymore.

5.4.8. Error messages

TBW. TBW.

TBW. TBW.

5.5. Deglitching the Timeline Data

5.5.1. Module Description

The *First Level Deglitcing* Module removes glitches due to cosmic ray hits or other impulse-like events. The basic assumption is that the glitch signature is similar to a Dirac delta function. This process is composed of two steps: the first step implements a wavelet-based local regularity analysis to detect glitch signatures in the measured signal; the second step locally reconstructs a signal free of such glitch signatures.

The wavelet method for deglitching is described in detail in;

1. Ordenovic C., Surace C., Torresani B., Llebaria A., Detection of glitches and signal reconstruction using Hoelder and wavelet analysis, Statistical Methodology, 2008, Volume 5, Issue 4, 373-386

2. Ordenovic C., Surace C., Torresani B., Llebaria A., Baluteau J.P., Use of a local regularity analysis by a wavelet analysis for glitch detection, SPIE, 2005, 5909, 556-567

5.5.2. Input Data Products

PDT; — **Photometer Detector Timeline** The input PDT product contains timelines for each photometer detector for each observation building block.

5.5.3. Output Data Products

PDT; — **Spectrometer Detector Timeline** The output PDT product contains deglitched timelines for each spectrometer detector for each observation building block.

5.5.4. Input Calibration Products

No calibration data are needed.

5.5.5. How to use the *First Level Deglitching* Module

5.5.6. Parameter Options

The first level deglitching task employs a complex algorithm, based on a continuous wavelet transform with a Mexican Hat wavelet and subsequent complex processing for a local regularity analysis. A thorough understanding of this algorithm is required in order to make good choices when setting the involved parameters. Users are strongly discouraged from changing the default value of these parameters unless they have closely studied the extended documentation for this task.

scaleMin, scaleMax, and scaleInterval — scaleMin/scaleMax are the minimum/maximum values of the range of wavelet scales used for the linear fit to the Wavelet Transform Modulus Maxima Lines. scaleMin and scaleMax should both be positive integers and scaleMin should be less than scaleMaxe. The cur-

	rent best estimation for the Jiggle Pipeline are values of scaleMin = 1, scaleMax= 8 and scaleInterval = 5. scaleInterval defines how many wavelet scales are calculated for a range of one within the scale range. The size of the range defined by scaleMin and scaleMax times the value of scaleInterval is directly proportional to execution time. Both should therefore be kept at the minimum required to accurately determine the Holder index.
holderMax and holderMin	holderMax/hmin are the minimum/maximum values of a real data type of the range of slopes which are interpreted as glitch indicators. The range should include -1 and holderMax should be larger than hmin. The current best estimation for the Jiggle Pipeline are values of holderMax = -0.1 and holderMin = -1.6. The range defined by holderMax and hmin allows to select between a more sensitive glitch detection to detect even weak glitches, or a more conservative glitch detection to avoid false positives, i.e. samples that are flagged as glitches which are arguably due to other causes.
correlationThreshold	This parameter relates to the square of the correlation coefficient of the linear fit to the Wavelet Transform Modulus Maxima Line. It sets a lower threshold of the correlation to enforce a minimum goodness of the fit to the data. This real number should be between 0 and 1, and is usually selected closer to 1. A value of 0 removes any selection criterion within the algorithm for goodness of fit. A value of 1 will make the criterion so stringent that only a perfect fit will be accepted. The current best estimation for the Jiggle Pipeline are values of correlationThreshold = 0.6. The value selected for correlationThreshold allows to select between a more sensitive glitch detection to detect even weak glitches, or a more conservative glitch detection to avoid false positives, i.e. samples that are flagged as glitches which are arguably due to other causes.

5.5.7. Examples

```
IA>> from herschel.spire.ia.pipeline.common.deglitch import DeglitchingTask
IA>>
IA>> deglitch=DeglitchingTask()
IA>> pdt=deglitch(pdt,
                  scaleMin=1,
     scaleMax=8,
     scaleInterval=5,
     holderMin=-1.6,
     holderMax=-0.1,
     correlationThreshold=0.6)
IA>>
```

5.5.8. Error messages

severe :unable to store output product. The task failed to write the output product.

warning :Unable to perform reconstruction. The task failed to reconstruct signal samples.

severe :unable to perform wavelet decomposition.. The task failed to identify glitches because the signal could not be decomposed into its wavelet components.

5.6. Removing Electrical Crosstalk from the Timeline Data

5.6.1. Module Description

The *Electrical Cross Talk Correction* module removes electrical crosstalk from the Detector Timelines. The module multiplies a crosstalk removal matrix with the signal vector for each time sample and returns Detector Timelines which are corrected for electrical crosstalk.

5.6.2. Input Data Products

PDT **Photometer Detector Timeline** The input PDT product contains timelines for each detector for each observation building block.

5.6.3. Output Data Products

PDT **Photometer Detector Timeline** The output PDT product contains timelines for each detector for each observation building block with the signal corrected for electrical crosstalk.

5.6.4. Input Calibration Products

SCalElecCross **Electrical Crosstalk Matrix** There are three electrical crosstalk matrices one for each of the SPIRE photometer arays. They contain entries for the electrical crosstalk between the detectors of each one of the two detector arrays. In the absence of any crosstalk, the matrices contain unity elements in the diagonals and zero elements anywhere else.

5.6.5. How to use the *Remove Electrical Crosstalk* Module

It should be noted that, prior to launch, no electrical crosstalk was observed. During the fabrication of the warm read-out electronics, spot-checking showed extremely low levels of electrical crosstalk. An additional analysis was performed for the detector arrays of the SPIRE imaging photometer using data from the ground-based test campaigns. This analysis checked whether the random instances of detectors absorbing ionizing radiation lead to reproducible signals in other detectors. Again, this analysis has shown no measurable evidence for electrical crosstalk. Close examination of the post-launch performance will be required to decide whether the detector arrays suffer significant electrical crosstalk. Until then, this module simply multiplies the detector timelines by the identity matrix.

5.6.6. Parameter Options

N/A. There are no options for this module.

5.6.7. Examples

Assuming that `pdt` is an PDT; product and `obs` an observation context:

```
pdt = elecCrossCorrection(pdt, elecCross=obs.calibration.phot.elecCross)
```

5.6.8. Error messages

TBW. TBW.

TBW. TBW.

5.7. Converting the Detector Timelines into Flux Units

5.7.1. Module Description

The *Flux Conversion* Module applies a correction for the nonlinear bolometer responsivity to a photometer detector voltage time line V (i.e., PDT) by integrating the function, f(V) = K1 + K2/(V-K3), from Vo to V, where the parameters Vo, K1, K2, and K3 are given for each and every detector channel in a calibration product (`SCalPhotFluxConv()`. The result is an in-beam flux density timeline.

5.7.2. Input Data Products

PDT	**Photometer Detector Timeline** Photometer Detector Timeline (PDT) with bolometer signal in electrical units (i.e. Volts)

5.7.3. Output Data Products

PDT	**Photometer Detector Timeline** Photometer Detector Timeline (PDT) product with bolometer signal in units of Jy

5.7.4. Input Calibration Products

ScalPhotFLuxConv	**Photometer Flux Conversion and Non-linearity Correction Coefficients** contains, for each bolometer channel, V0, K1, K2, K3 and their uncertainties, as well as Vmin and Vmax, the calibrated limiting bolometer voltages.

5.7.5. How to use the *Flux Conversion* Module

5.7.6. Parameter Options

There are no parameter options for this module.

5.7.7. Examples

Jython Usage Example: Assuming a detector timeline of SPIRE photometer data processed to Level 0.5, with bolometer signal data in units of volts `PDT` and a calibration product, `fluxConv`.

```
outputfluxproduct = photFluxConversion(pdt,fluxConv=obs.calibration.phot.fluxConv)
```

5.7.8. Error messages

Error messages generated:

If the calculations result in taking the log of a negative number, the exception will be noted and the point generating the exception will be flagged in the output product mask.

5.8. De-Modulating the Timeline Data

5.8.1. Module Description

The *DeModulation* Module computes the amplitude of the modulation of the photometer dued to the motion of the chopper. There is one point per chopper cycle per bolometer.

5.8.2. Input Observational Data Products

PPP — **Pointed Photometer Product** A Pointed Photometer Product timeline as described in http://www.spire.rl.ac.uk/icc/product_definitions/

CJT — **Chop Jiggle Timeline** It contains the start time and end time and the id of each consecutive jiggle map position. In addition, for debugging purpose, the converted values Y,Z of each jiggle map position is also stored. For the definition see: http://www.spire.rl.ac.uk/icc/product_definitions/

5.8.3. Output Data Products

DPPP — **Demodulated Photometer Product** The module outputs a Demodulated Photometer Product desscribed by http://www.spire.rl.ac.uk/icc/product_definitions/.

5.8.4. Input Calibration Products Required

TBW — **not yet defined** The calibration product is not yet defined. It must contain two time contants, a delay time and a transition time. We don't know yet if these contants depend will vary for each bolometer or for each array of bolometers.

5.8.5. How to use the *Demodulation* Module

The Demodulation takes a `PointedPhotTimeline` object as input. This Pointed Photometer Product (PPP) contains the signal and astrometry timeline of each bolometer.

The Demodulation also takes a `CJT` object as input. This Chop Jiggle Product (CJT) contains the mirror angles timeline.

The Demodulation task output is `DemodPhotProduct` Photometer Demodulated Detector Product with one point per bolometer and per chopper cycle.

5.8.6. Parameter options

Users can set the following parameters of the Demodulation module:

5.8.7. Examples

Assuming that `ppt` is a Pointed Photometer Product and `cjt` is a CJT products:

```
demodulate=DemodulateTask()
dpp = demodulate(ppt=ppt, cjt=cjt)
```

Note that, since the calibration product for the time constants does not exist, default values are used.

5.8.8. Error messages

5.9. Averaging the Data of All Jiggle Positions (and Second Level Deglitching)

5.9.1. Module Description

The *Second Level Deglitching and Averaging* Module

5.9.2. Input Data Products

Input Product name here **Input Product name here** and description to follow here

5.9.3. Output Data Products

Output Product Here **Output Product Name Here** and description to follow here

5.9.4. Input Calibration Products

Calibration Product Name Here **Calibration Product Name Here** and description to follow here

5.9.5. How to use the *Second Level Deglitching and Averaging* Module

5.9.6. Parameter Options

Users can set the following parameters of the module:

5.9.7. Examples

5.9.8. Error messages

TBW. TBW.

TBW. TBW.

5.10. De-Nodding Observations

5.10.1. Module Description

The *De-Nod* module de-nods data and averages multiple nodding cycles to return the signal per detector.

5.10.2. Input Observational Data Products

The module will take as input one, two or four Demodulated Photometer Products (DPP).

No product definition document available yet.

5.10.3. Output Data Products

The module outputs a Pointed Photometer Product (PPP).

See the Herschel Products Definition Document or alternatively http://www.spire.rl.ac.uk/icc/product_definitions/.

5.10.4. Input Calibration Products Required

No calibration products required.

5.10.5. How to use the *Denodding* Module

The module takes a `DenodInput` object as input. This is just a container for one, two or four Demodulated Photometer Products (DPP).

The module is made of two tasks, Denodding and NodAveraging. There is no wrapper task available for this as it currently stands, so they will have to be called explicitly.

The Denodding task only takes one DenodInput as input and gives a Pointed Photometer Product (PPP) as output. The task must be executed for every set of nod positions (either A-B or A-B-B-A cycles).

The NodAveraging task takes an array of PPPs as input and gives a single PPP as output. Two optional parameters are available:

- **operator:** string parameter, can be one of `mean`, `median` or `weighted mean`. Used to choose the averaging operator. Default is `mean`.
- **deglitch:** integer operator, used to toggle removal of outliers by setting it to either 0 or 1. Note that deglitching is not implemented at the moment. Default is 1.

5.10.6. Parameter options

Users can set the following parameters of the Denodding module:

5.10.7. Examples

Assuming that `dpp1` and `dpp2` are two DPP products:

```
denodInput = DenodInput()
denodInput.addProduct(dpp1)
denodInput.addProduct(dpp2)
denoddedPpp = DeNodding()(input=denodInput)
averagedPpp = nodAverage()(input=[denoddedPpp])
```

Note that, since only one PPP is given as input to Averaging, the output product will contain the same values.

5.10.8. Error messages

"Error in DeNodding task: no ADTs have been loaded as input products by DeNodInput"

"The two ADTs are not an A and a B position and can't be denodded"

"ADT4 does not appear to contain any nod A or B positions."

5.11. Averaging the Data Taken from Different Nod Cycles

5.11.1. Module Description

The *Average Nod Cycles* Module

5.11.2. Input Data Products

Input Product name here **Input Product name here** and description to follow here

5.11.3. Output Data Products

Output Product Here **Output Product Name Here** and description to follow here

5.11.4. Input Calibration Products

Calibration Product Name Here **Calibration Product Name Here** and description to follow here

5.11.5. How to use the *Average Nod Cycles* Module

5.11.6. Parameter Options

Users can set the following parameters of the module:

5.11.7. Examples

5.11.8. Error messages

TBW. TBW.

TBW. TBW.

5.12. Removing the Effects of Optical Crosstalk from the Data

5.12.1. Module Description

The *Optical Crosstalk Removal* Module

5.12.2. Input Data Products

Input Product name here **Input Product name here** and description to follow here

5.12.3. Output Data Products

Output Product Here **Output Product Name Here** and description to follow here

5.12.4. Input Calibration Products

Calibration Product Name Here **Calibration Product Name Here** and description to follow here

5.12.5. How to use the *Optical Crosstalk Removal* Module

5.12.6. Parameter Options

Users can set the following parameters of the module:

5.12.7. Examples

```
pdt = photOptCrossCorrection(pdt, optCross=obs.calibration.phot.optCross)
```

5.12.8. Error messages

TBW. TBW.

TBW. TBW.

5.13. Converting the On-Board Time to International Time Standards

5.13.1. Module Description

The *Time Correction* Module

5.13.2. Input Data Products

Input Product name here **Input Product name here** and description to follow here

5.13.3. Output Data Products

Output Product Here **Output Product Name Here** and description to follow here

5.13.4. Input Calibration Products

Calibration Product Name Here **Calibration Product Name Here** and description to follow here

5.13.5. How to use the *Time Correction* Module

5.13.6. Parameter Options

Users can set the following parameters of the module:

5.13.7. Examples

5.13.8. Error messages

TBW. TBW.

TBW. TBW.

5.14. Creating Image Maps from the Jiggle Observations

5.14.1. Module Description

This module provides the map-making routines used by the SPIRE photometer pipeline. Jiggle mode observations are processed after denodding and averaging and the resulting map is obtained using the task `NaiveApppMapperTask`. All output maps are North-oriented.

5.14.2. Input Observational Data Products

APPP — **Averaged Pointed Photometer Product** The `NaiveApppMapperTask` takes one Averaged Pointed Photometer Product (APPP) as input. For a definition, see:

http://www.spire.rl.ac.uk/icc/product_definitions/

5.14.3. Output Data Products

All map-making tasks returns the map associated to the bolometer array specified by the user. The map is stored as a `SimpleImage`. For a definition, see:

ftp://ftp.rssd.esa.int/pub/HERSCHEL/csdt/releases/doc/api/herschel/ia/dataset/image/SimpleImage.html

5.14.4. Input Calibration Products Required

there are no calibration files required for the Jiggle Map `NaiveApppMapperTask`

5.14.5. How to use the *Map Making* Module

The module contains 3 mappers: 2 for the scan mode (`NaiveScanMapperTask` and `MadScanMapperTask`) and 1 for the jiggle mode (`NaiveApppMapperTask`).

The input parameters are a context of Photometer Detector Timelines (`PointedProduct`) for the scan mode and an Averaged PointedPhotometer Product for the jiggle mode.

The user should set the `array` keyword to "PSW", "PMW" or "PLW" to select the bolometer array to be processed. The optional keyword `maxmemory` can be used to increase the memory to be allocated for map-making and avoid slowing down by limiting I/O.

The mappers only compute one map at a time, for the bolometer array specifed by the user. The output map is stored as a `SimpleImage` product, for which convenient visualisation tools are available (see examples below) within HCSS.

The mappers use temporary files to store the image coordinates (X,Y). `IOException` will be thrown is these temporary files can not be written or read. Out of memory exception can occur if the size of the map is too large (because of limited resource or incorrect astrometry)

5.14.6. Parameter options

Users can set the following parameters of the Map Making module:

array — **Name of the bolometer array to be processed** The value of this parameter can be "PSW", "PMW" or "PLW". If not set, the default bolometer array is assumed to be "PSW".

resolution — **Pixel size of the output map** The value of this parameter is in arcseconds. If not set, the default values are 6", 10" and 14 for the PSW, PMW and PLW bolometer array.

maxmemory — **Maximum memory in bytes to be used to hold the pointing matrix and the timeline in memory.** If the allocated memory isn't large enough, processing time may be I/O limited. By default, 1GB is assumed.

5.14.7. Examples

Given an Averaged Photometer Pointing Product, the jiggle map is simply obtained in the following way.

```
IA>> from herschel.spire.ia.pipeline.phot.scanmap import NaiveApppMapperTask
IA>> mapper = NaiveApppMapperTask()
IA>> psw = mapper(appp, array='PSW')
IA>> Display(psw)
```

5.14.8. Error messages

5.15. Calculating the Flux and Position of a Source

5.15.1. Module Description

The *Point Source Extraction* module gives the position and flux of a point source observed with a 7-point jiggle pattern. As an intermediate result, it gives the outcome of trying to fit a PSF to the signal of every detector.

5.15.2. Input Data Products

The module takes one Pointed Photometer Product (PPP) as input. For the product definition see:

http://www.spire.rl.ac.uk/icc/product_definitions/

5.15.3. Output Data Products

The final output of this module is a Jiggled Photometer Product. For the definition see:

http://www.spire.rl.ac.uk/icc/product_definitions/

An intermediate output, called Jiggle Point Source Fit Product (JPSFP), will be produced by the first task of the module. For the definition see:

http://www.spire.rl.ac.uk/icc/product_definitions/

5.15.4. Input Calibration Products Required

No calibration products required.

5.15.5. How to use the *Point Source Flux Density and Position* Module

There are two tasks making up this module, `PointSourceFit` and `SourceFlux`. The former takes a Pointed Photometer Product as input and tries to fit a PSF (currently just a two-dimensional symmetric Gaussian) to the signal of every detector. It outputs a Jiggle Point Source Fit Product. The input product is the only parameter.

The latter task takes the Jiggle Point Source Fit Product as input and produces a Jiggled Photometer Product with the position and flux of the point source, as measured in each of the three detector arrays. There are three optional input parameters, called `positivePixels`, `negativePixels1` and `negativePixels2`. They are `String` arrays taking the name of the pixels where the positive and the two negative images of the source are expected, for each array.

5.15.6. Parameter options

Users can set the following parameters of the Point Source module:

5.15.7. Examples

If `myPpp` is a Pointed Photometer Product with the data you want to analyse, the following example show the steps needed to obtain a Jiggled Photometer Product named `myJpp`:

```
myJpsfp = PointSourceFit()(input = myPpp)
myJpp = SourceFlux()(input = myJpsfp)
```

5.15.8. Error messages

Chapter 6. Processing Data with the Spectrometer Pipeline

6.1. Introduction

6.1.1. The Spectrometer Pipeline

A detailed description of the Spectrometer pipeline design can be found in two technical papers:

Fulton, T.R., Naylor, D.A., Baluteau, J.-P., Griffin, M., Davis-Imhof, P., Swinyard, B.M., Lim, T., Surace, C., Clements, D., Panuzzo, P., Gastaud, R., Polehampton, E.T., Guest, S., Lu, N., Schwartz, A., Xu, K., The data processing pipeline for the Herschel SPIRE Imaging Fourier Transform Spectrometer. Proc. SPIE, Space Telescopes and Instrumentation: Optical, Infrared, and Millimeter 7010, (2008)

Abstract: We present the data processing pipeline to generate calibrated data products from the Spectral and Photometric Imaging Receiver (SPIRE) imaging Fourier Transform Spectrometer. The pipeline processes telemetry from SPIRE point source, jiggle- and raster-map observations, producing calibrated spectra in low-, medium-, high-, and mixed low- and highresolution modes. The spectrometer pipeline shares some elements with the SPIRE photometer pipeline, including the conversion of telemetry packets into data timelines and the calculation of bolometer voltages from the raw telemetry. We present the following fundamental processing steps unique to the spectrometer: temporal and spatial interpolation of the stage mechanism and detector data to create interferograms; apodization; Fourier transform, and creation of a hyperspectral data cube. We also describe the corrections for various instrumental effects including first- and secondlevel glitch identification and removal, correction of the effects due to the Herschel primary mirror and the spectrometer calibrator, interferogram baseline correction, channel fringe correction, temporal and spatial phase correction, non-linear response of the bolometers, variation of instrument performance across the focal plane arrays, and variation of spectral efficiency. Astronomical calibration is based on combinations of observations of standard astronomical sources and regions of space known to contain minimal emission.

Fulton, T.R., Baluteau, J.-P., Bendo, G., Benielli, D., Gastaud, R., Griffin, M., Guest, S., Imhof, P., Lim, T.L., Lu, N., Naylor, D.A., Panuzzo, P., Polehampton, E., Schwartz, A., Surace, C., Swinyard, B.M., Xu, K., The data processing pipelines for the Herschel/SPIRE Imaging Fourier Transform Spectrometer. Proc. SPIE, Space Telescopes and Instrumentation: Optical, Infrared, and Millimeter Wave 7731, (2010)

Abstract: We present an update to the data processing pipelines that generate calibrated spectral data products from the Spectral and Photometric Imaging Receiver (SPIRE), one of three scientific instruments onboard the European Space Agency's Herschel Space Observatory launched on 14 May 2009. The pipelines process telemetry from SPIRE's imaging Fourier Transform Spectrometer (FTS) taken in point source, jiggle- and raster-map observing modes, producing calibrated spectra in low-, medium-, high-, and mixed low- and high-spectral resolution. While the order and algorithms of the data processing modules in the spectrometer pipelines remain for the most part unchanged compared to their pre-launch status, some improvements and optimizations have been realized through the analysis of data from the performance verification and science demonstration phases of the mission. The data processing pipelines for the SPIRE FTS as of the beginning of the routine phase of the Herschel mission are presented in their entirety, with more detailed descriptions reserved for those elements that have changed since launch, in particular the first- and second-level correction steps for glitches, the step that corrects for clipped samples, and the process by which Level-1 spectral data are converted to Level-2 products. In addition, we discuss some of the challenging aspects still faced by the automated processing pipelines, such as the removal of the contributions from the Herschel telescope and SPIRE instrument, and the relative spectral response correction and flux conversion steps.

Figure 6.1. Flowchart for the Spectrometer Pipeline.

6.1.2. Multiple Calibration Products

It is important to use the correct calibration data when processing data. In order to make this easier, all data processing modules implement the following rule:

If a module can accept three or more calibration products, it provides an option whereby it accepts the calibration context as a whole. If any calibration product is passed to a module explicitly in addition to the calibration context, the explicit calibration product takes precedence. This applies, for example, to *Create Interferogram*.

6.1.3. Deep Copy of Data Products

Users should note that the data processing pipeline overwrites the input product whenever possible to reduce the amount of required computer memory. If the original input product is to be kept, then specific care needs to be taken. For example, create a deep copy of the input product before it is passed to the module. Sample code is given below for the three different types of input products for spectrometer pipeline modules.

Spectrometer Detector Timeline

```
sdtOriginal = DetectorTimeline(sdt)
sdt = clippingCorrection(sdt)
print sdtOriginal.diff(sdt) # to see the differences
```

Spectrometer Detector Interferogram

```
sdiOriginal = sdi.copy()
sdi=deglitchIfgm(sdi=sdi, deglitchType="MAD_WINDOW", \
                     windowSize=33)
print sdiOriginal.diff(sdi) # to see the differences
```

Spectrometer Detector Spectrum

```
asdsOriginal = asds.copy()
# myFluxConv must be a valid flux conversion calibration product
asds = specFluxConversion(sds=asds, fluxConv=myFluxConv)
print asdsOriginal.diff(asds) # to see the differences
```

6.2. Correcting the Detector Timelines for Electrical Crosstalk

6.2.1. Module Description

The *Electrical Cross Talk Correction* module removes electrical crosstalk from the Spectrometer Detector Timelines (SDT). The task multiplies a crosstalk removal matrix with the signal vector for each instant in time and returns an SDT which is corrected for electrical crosstalk.

6.2.2. Input Data Products

SDT — **Spectrometer Detector Timeline** The input SDT product contains timelines for each spectrometer detector for each observational building block.

6.2.3. Output Data Products

SDT — **Spectrometer Detector Timeline** The output SDT product contains timelines for each spectrometer detector for each observation building block with the signal corrected for electrical crosstalk.

6.2.4. Input Calibration Products

elecCross — **Electrical Crosstalk Matrix** The two electrical crosstalk matrices for the SPIRE spectrometer contain entries for the electrical crosstalk between detectors from the two arrays SLW and SSW. The calibration table does not allow for crosstalk between detectors in different arrays. In the absence of any crosstalk, the matrices contain unity elements in the diagonals and zero elements anywhere else.

6.2.5. How to use the *Electrical Cross Talk Correction* Module

Prior to launch, no electrical crosstalk was observed for the two spectrometer arrays. Close examination of the in-flight performance will be required to decide whether the detector arrays suffer significant electrical crosstalk. Until then, this processing task is a placeholder.

6.2.6. Parameter Options

The Electrical Crosstalk Removal module does not use any parameters that can be controlled by the user during run-time.

6.2.7. Examples

Assuming that `sdt` is an SDT product and `obs` an observation context:

```
from herschel.spire.ia.pipeline.common.eleccross import ElecCrossCorrectionTask
sdt = elecCrossCorrection(sdt, elecCross=obs.calibration.spec.elecCross)
```

6.2.8. Error messages

error: The input product is not a Photometer or Spectrometer Detector timeline The electrical crosstalk removal task requires a detector timeline product as input (either for the photometer or the spectrometer). The module cannot operate on a product of any other type.

error: The input product is already corrected. The electrical crosstalk removal task should only be applied once during data processing. An error is thrown if the module is applied more than once and the module is not executed.

6.3. Deglitching the Detector Timelines

6.3.1. Module Description

The *Wavelet Timeline Deglitching* Module removes glitches due to cosmic ray hits or other impulse-like events. The fundamental assumption is that the glitch signature is similar to a Dirac delta

function. This process is composed of two steps: the first step implements a wavelet-based local regularity analysis to detect glitch signatures in the measured signal; the second step locally reconstructs the signal.

The wavelet method for deglitching is described in detail in;

1. Ordenovic C., Surace C., Torresani B., Llebaria A., Detection of glitches and signal reconstruction using Hoelder and wavelet analysis, Statistical Methodology, 2008, Volume 5, Issue 4, 373-386

2. Ordenovic C., Surace C., Torresani B., Llebaria A., Baluteau J.P., Use of a local regularity analysis by a wavelet analysis for glitch detection, SPIE, 2005, 5909, 556-567

6.3.2. Input Data Products

SDT — **Spectrometer Detector Timeline** The input SDT product contains timelines for each spectrometer detector for each observation building block.

6.3.3. Output Data Products

SDT — **Spectrometer Detector Timeline** The output SDT product contains deglitched timelines for each spectrometer detector for each observation building block.

6.3.4. Input Calibration Data Products

No calibration data are needed.

6.3.5. How to use the *Wavelet Timeline Deglitcher* Module

6.3.6. Parameter Options

The first level deglitching task employs an algorithm which is based on a continuous wavelet transform with a Mexican Hat wavelet and subsequent processing for a local regularity analysis. A thorough understanding of this algorithm is required in order to make good choices when setting the involved parameters. Users are strongly discouraged from changing the default value of these parameters unless they have closely studied the detailed technical documentation for this task.

scaleMin, scaleMax, and scaleInterval (Optional) — scaleMin/scaleMax are the minimum/maximum values of the range of wavelet scales used for the linear fit to the Wavelet Transform Modulus Maxima Lines. scaleMin and scaleMax should both be positive integers and scaleMin should be less than scaleMax. For example, a feasible range for the practical purposes of the SPIRE FTS is defined by the values 1 and 8 for scaleMin and scaleMax. scaleInterval defines how many wavelet scales are calculated for a range of one within the scale range. The size of the range defined by scaleMin and scaleMax times the value of scaleInterval is directly proportional to execution time. The range and number of elements in an interval should therefore be kept at the minimum required to accurately determine the Holder index.

holderMin and holderMax (Optional) — holderMin/holderMax are the minimum/maximum values of the range of slopes which are interpreted as glitch indicators. The range should include -1 and holderMin should be smaller than holderMax. For example, a feasible range for the practical purposes of the SPIRE FTS is defined by the values -1.4 and -0.6 for holderMin and holderMax. The range defined by holderMin and holderMax allows to se-

	lect between a more sensitive glitch detection to detect even weak glitches, or a more conservative glitch detection to avoid false positives, i.e. samples that are flagged as glitches which are arguably due to the nominal operation of the instrument.
correlationThreshold (Optional)	This parameter relates to the square of the correlation coefficient of the linear fit to the Wavelet Transform Modulus Maxima Line. It sets a lower threshold of the correlation to enforce a minimum goodness of the fit to the data. This number should be between 0 and 1, and is usually selected closer to 1. A value of 0 removes any selection criterion within the algorithm for goodness of fit. A value of 1 will make the criterion so stringent that only a perfect fit will be accepted. A feasible value for the practical purposes of the SPIRE FTS is 0.85. The value selected for correlationThreshold allows to select between a more sensitive glitch detection to detect even weak glitches, or a more conservative glitch detection to avoid false positives, i.e. samples flagged as glitches, which are arguably due to the nominal operation of the instrument.
correctGlitches (Optional)	The user can select to correct the identified glitches or not. If the value of this parameter is False then the task will still flag glitches by setting the GLITCH_FIRST_LEVEL mask bit, but it will not change the detector signal. By default, this parameter is set to True.
optionReconstruction (Optional)	Three options are available to change detector samples flagged as glitches. They are "polynomialFitting", "linearInterpolation", "wavelette". It is strongly recommended to select the option of polynomial fitting. It has been shown to give the best results. The option to perform an inverse wavelet transform is still under development.
degreePoly (Optional)	If the option of polynomial fitting has been selected, then it is also possible to set the degree of the polynomial. Tests have shown that the deglitcher gives good results when using a polynomial of degree 6.
fitPoints (Optional)	Reconstruction is based on a number of points before and after the glitch which are used by the fitting routines to replace the detector samples. This number can be adjusted. Tests have shown that the deglitcher gives good results when fitting to a total of 8 data points.
reconstructionPointsBefore (Optional)	One sample will be identified as the peak of the glitch. This option allows to control how many samples (up to 30) will be changed before the peak of the glitch. By default, 2 samples before the peak of the glitch will be changed.
reconstructionPointsAfter (Optional)	One sample will be identified as the peak of the glitch. This option allows to control how many samples (up to 30) will be changed after the peak of the glitch. By default, 3 samples after the peak of the glitch will be changed.

6.3.7. Examples

```
from herschel.spire.ia.pipeline.common.deglitch import WaveletDeglitcherTask
sdt=deglitchTimeline(sdt, \
             scaleMin=1, \
             scaleMax=8, \
             scaleInterval=5, \
             holderMin=-1.4, \
             holderMax=-0.6, \
             correlationThreshold=0.85, \
```

```
                correctGlitches=Boolean.TRUE, \
                optionReconstruction="polynomialFitting", \
                degreePoly=6, \
                fitPoints=8, \
                reconstructionPointsBefore=2, \
                reconstructionPointsAfter=3)
```

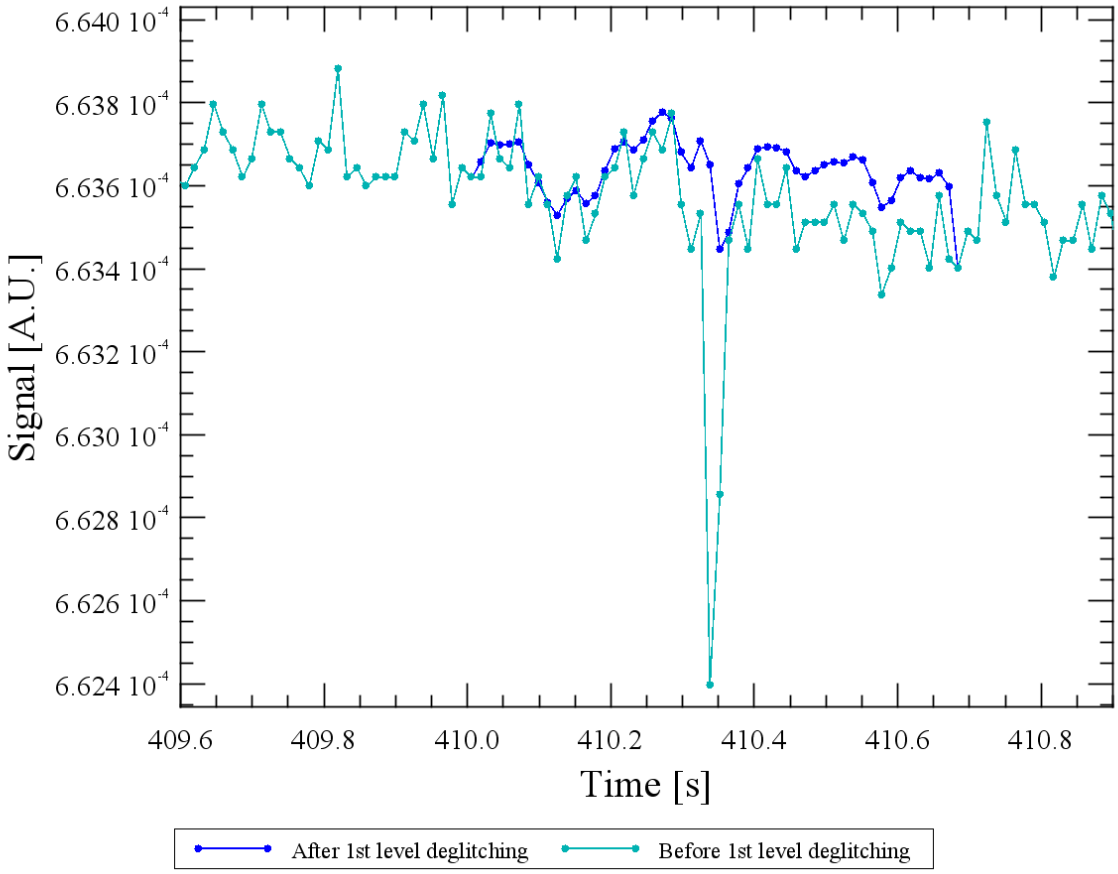

Figure 6.2. An example for glitch removal in a the detector timeline from the SPIRE spectrometer array. An original detector timeline (green) from SLWC1 is shown after removal of the glitch (blue).

6.3.8. Error messages

There are no specific error messages defined for this task.

6.4. Applying the Non-Linearity Correction to the Data

6.4.1. Module Description

The *Non-Linearity Correction* module applies a correction for the nonlinear bolometer responsivity to a spectrometer detector voltage timeline V (SDT) by integrating the function, f(V) = K1 + K2/(V-K3), from Vo to V, where the parameters Vo, K1, K2, and K3 are given for each and every detector

channel in a calibration product (SCalSpecNonLinCorr). The result is a voltage timeline SDT where the voltage is directly proportional to the power incident on a detector.

6.4.2. Input Data Products

SDT — **Spectrometer Detector Timeline** The input SDT with its bolometer signal converted to electrical units (e.g. V).

6.4.3. Output Data Products

SDT — **Spectrometer Detector Timeline** The output SDT product with bolometer signal in units of Volts corrected for any non-linearity, ideally being linearly proportional to incident radiation.

6.4.4. Input Calibration Products

nonLinCorr — **Spectrometer Non-Linearity Correction** contains, for each bolometer channel, V0, K1, K2, K3 and their uncertainties, as well as Vmin and Vmax, the limit bolometer voltages for which the calibration is valid.

6.4.5. How to use the *Non-Linearity Correction* Module

6.4.6. Parameter Options

The Nonlinearity Correction module does not use any parameters that can be controlled by the user at run-time.

6.4.7. Examples

Assuming a detector timeline of SPIRE spectrometer data processed to Level 0.5, with bolometer signal data in units of volts `sdt` and `obs` is an observation context:

```
from herschel.spire.ia.pipline.spec.nonlin import SpecNonLinearityCorrectionTask
# get the bias mode (biasMode) from the SDT
biasMode = sdt.meta["biasMode"].value
# get the observation start date from the SDT
date = sdt.startDate
# get the correct edition of the calibration product
nonLinCorr = obs.calibration.spec.nonLinCorrList.getProduct(biasMode, date)

linearizedSdt = specNonLinearityCorrection(sdt, nonLinCorr=nonLinCorr)
```

6.4.8. Error messages

error: The input product is not a Photometer or Spectrometer Detector timeline. This module can only process a detector timeline product, either from the photometer or the spectrometer.

error: the input product is already corrected. This module should only be applied once during data processing. An error is thrown if the module is applied more than once.

error: controlStamp parameter is not specified in calibration product.

Error messages generated: If the calculations result in taking the log of a negative number, the exception will be noted and the point generating the exception will be flagged in the output product mask.

6.5. Removing Correlated Noise due to bath temperature fluctuations from the Data

6.5.1. Module Description

The *Bath Temperature Fluctuation Correction* module subtracts low frequency (< 1Hz) noise, caused by variations of the bath temperature of the detector arrays. The module is based on the empirical correlations between detectors and thermistors (or, in case of high bias voltages when thermistors are saturated, the correlations between detectors and dark pixels). It is important to note that this module works in conjunction with and always **after** the nonlinearity correction module.

6.5.2. Input Data Products

SDT — **Spectrometer Detector Timeline** The SDT with the bolometer signal converted to electrical units (e.g. V) directly proportional to the incident power.

6.5.3. Output Data Products

SDT — **Spectrometer Detector Timeline** SDT in the units of V, corrected for temperature drift.

6.5.4. Input Calibration Products

tempDriftCorrList — **Temperature Drift Correction Parameter Table** A product containing for each detector array, a thermistor selection flag, the base voltage values for each thermistor, the errors for those values, and a validity calibration flag for those values. It contains for each bolometer channel, correction coefficients A1, A1 error, B1, B1 error, AB1 flag (all for thermistor 1) and similar values for thermistor 2. Different editions for this calibration product exist depending on the bias mode (nominal or bright) and the time when the observation was performed.

6.5.5. How to use the *Bath Temperature Fluctuation Correction* Module

6.5.6. Parameter Options

timeSpan (Optional) — This is the smoothing timespan in seconds when averaging the signal from the thermistors. The default value is 5 (set within the task itself).

6.5.7. Examples

Assuming a detector timeline of SPIRE spectrometer data, `sdt` with bolometer signal data in units of V and processed by the spectrometer Non-Linearity Correction Module and an observation context `obs`:

```
   from herschel.spire.ia.pipepline.common.tempdrift import
TemperatureDriftCorrectionTask
   sdt=temperatureDriftCorrection(sdt, \
    tempDriftCorr=obs.calibration.spec.tempDriftCorrList.getProduct( \
    sdt.meta["biasMode"].value, sdt.startDate), timeSpan=5)
```

6.5.8. Error messages

error: the input product is not a Photometer or Spectrometer Detector timeline. This module can only process a detector timeline product, either from the photometer or the spectrometer.

error: the input product is already corrected. This module should only be applied once during data processing. An error is thrown if the module is applied more than once.

error: biasMode parameter is incorrectly specified in calibration product.

error: biasMode parameter is not specified in calibration product.

error: PswThermistorSelectValue must be one of T1, T2,or T1+T2.

error: PlwThermistorSelectValue must be one of T1, T2,or T1+T2.

error: PmwThermistorSelectValue must be one of T1, T2,or T1+T2.

error: controlStamp parameter is not specified in calibration product. User specified an obsolete version of the calibration product. (Calibration product should be from HCSS Build Version 1.1 OR LATER.)

error: controlStamp parameter is not specified in input product. NonLinearityCorrectionTask must be run first to generate control stamp in input metadata.

error: "xx" parameter is not specified or is incorrectly specified in for array "SSW/SLW" of the calibration product.

error: timeSpan parameter is not specified in calibration product.

error: VT02 parameter is not specified in calibration product. .

error: VT02 parameter is not specified in calibration product..

error: VT01Flag parameter is not specified in calibration product.

error: VT02Flag parameter is not specified in calibration product.

error: timeSpan parameter is not specified in calibration product.

error: biasSwitch failure - program logic error.

error: More than: "xx" percent of samples in the T1 thermistor timeline are marked as truncated points for the detector array: "yy"

error: More than: "xx" percent of samples in the DP1 dark pixel timeline are marked as truncated points for the detector array: "yy"

error: More than: "xx" percent of samples in the T2 thermistor timeline are marked as truncated points for the detector array: "yy"

error: More than: "xx" percent of samples in the DP2 dark pixel timeline are marked as truncated points for the detector array: "yy"

error: More than: "xx" percent of samples in both the T1 and T2 thermistor timelines are marked as truncated points for the detector array: "yy"

error: More than: "xx" percent of samples in both the DP1 and DP2 dark pixel timelines are marked as truncated points for the detector array: "yy"

6.6. Correcting for Clipped Data

6.6.1. Module Description

The 16-bit Analogue-to-Digital Converter (ADC) to read the detector voltages into SPIRE's on-board digital processing unit imposes a minimum (0) and maximum (65535) value to the detector readings. The two extreme samples are flagged as truncated by the Check ADC Flags and Truncation module at an earlier stage of the pipeline. The clipping correction module reconstructs those samples of the spectrometer detector timeline (SDT) which are flagged as truncated. Samples are reconstructed by using a polynomial fitting routine for each data range affected by truncation.

6.6.2. Input Data Products

SDT — **Spectrometer Detector Timeline** The input SDT product contains timelines for each spectrometer detector for each observation building block.

6.6.3. Output Data Products

SDT — **Spectrometer Detector Timeline** The output SDT product contains timelines for each spectrometer detector for each observation building block.

6.6.4. Input Calibration Products

No calibration data are needed.

6.6.5. How to use the *Clipping Correction* Module

6.6.6. Parameter Options

The Clipping Correction task employs a simple algorithm, based on fitting a number of data samples before and after the range of clipped samples with a high order polynomial function (currently fixed at 8). Users can change the default value of 5 for the number of data samples used for the fit. However, they are strongly discouraged from changing the default value to ensure an effective execution of the module.

fitPoints (Optional) — The parameter "fitPoints" defines the number of points used for the polynomial fitting to either side of a truncated data range. Overall, 2 x "fitPoints" will be used for the polynomial fitting. "fitPoints" must be an integer greater than 3 and less than 10.

6.6.7. Examples

```
from herschel.spire.ia.pipeline.spec.clip import ClippingCorrectionTask
```

```
sdt=clippingCorrection(input=sdt, fitPoints=5)
```

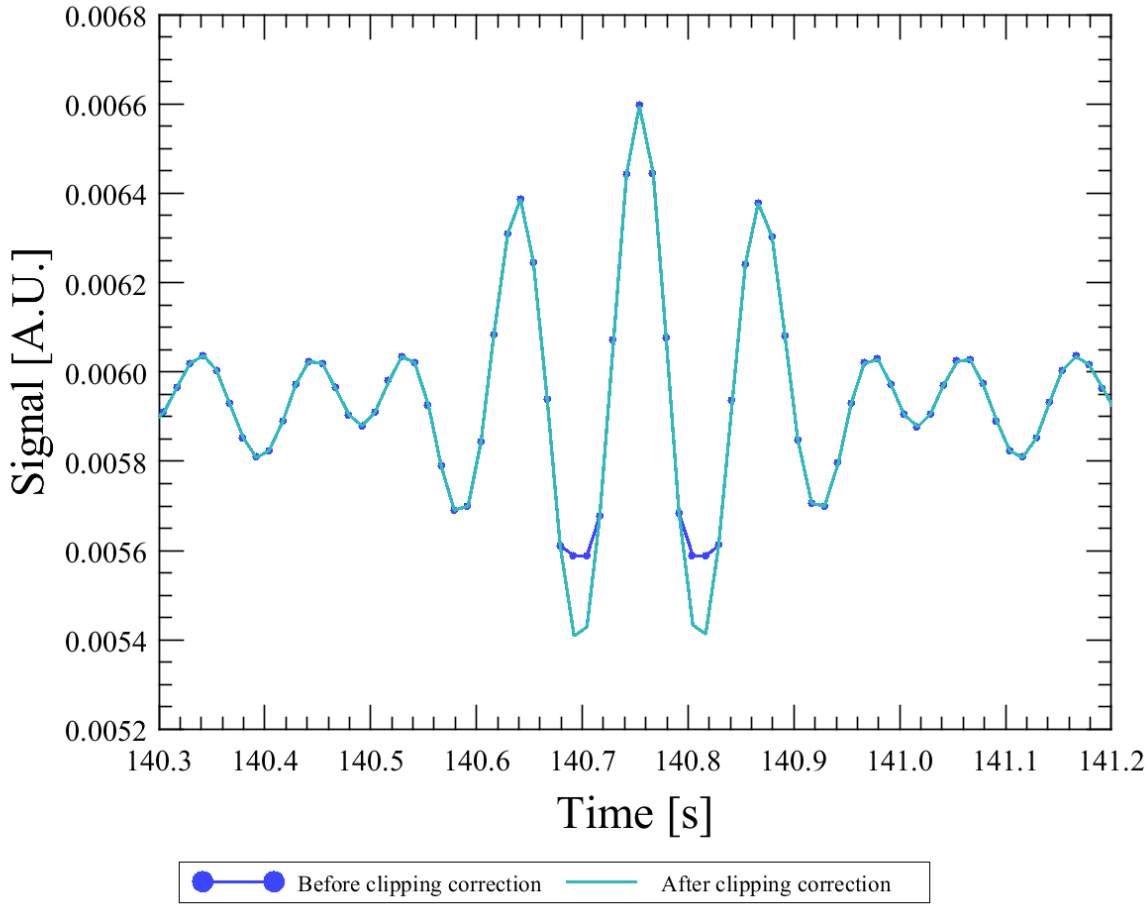

Figure 6.3. Detector Timeline before and after correcting for clipped data.

6.6.8. Error messages

error: Number of fitting points must be between 4 and 9.

6.7. Correcting for Time Domain Phase Shifts

6.7.1. Module Description

The *Time Domain Phase Correction* task corrects for the filtering effects due to the read-out electronics and the thermal behaviour of the bolometer detectors.

The SPIRE spectrometer detector chain contains a 6-pole Bessel low pass filter (LPF) and an additional RC LPF. In addition to the electronic LPF, the thermal behaviour of the SPIRE bolometers is modeled as an RC LPF with a detector-specific time constant, τ. The LPFs will affect the magnitude of the signal recorded by the SPIRE detectors and induce a phase shift to the recorded signal. The phase shift is characterized by comparing forward and reverse interferograms and a thermal detector-specific

time constant τ is derived. The thermal time constants are retrieved from a calibration product and time domain phase correction functions are computed accordingly. The measured detector timelines are corrected by a convolution with the derived time domain phase correction function. The edges of the timelines, invalidated by the convolution operation are truncated.

6.7.2. Input Observational Data Products

SDT	**Spectrometer Detector Timeline** The input SDT product contains timelines for each spectrometer detector for each observation building block.
NHKT (Optional)	**Nominal Housekeeping Timeline** This product contains the timelines for the nominal housekeeping telemetry parameters.

6.7.3. Output Data Products

SDT	**Spectrometer Detector Timeline** The output SDT product contains timelines for each spectrometer detector for each observation building block, corrected for the low pass filtering.

6.7.4. Input Calibration Data Products

lpfPar	**Spectrometer Low Pass Filter Parameters** This calibration product contains the parameters of the components of the read-out electronics of the SPIRE spectrometer.
chanTimeConst	**Spectrometer Detector Time Constants** This calibration product contains the thermal relaxation time constant τ per detector. The time constants range between 2.75 and 12.5 ms.
phaseCorrLim (Optional)	**Phase Correction Limits** This optional calibration product indicates where the pipeline expects the spectral SNR to be at its maximum in order to be suitable for a fit to the phase. The spectral ranges are given in units of cm^{-1} and correlate roughly to the measured spectral bands for each spectrometer detector.

6.7.5. How to use the *Time Domain Phase Correction* Module

6.7.6. Control parameters

The Time Domain Phase Correction module does not use any parameters that can be controlled by the user during run-time.

6.7.7. Examples

Assuming `sdt` is a valid Spectrometer Detector Timeline product and `obs` an observation context, time domain phase correction can be applied as follows:

```
from herschel.spire.ia.pipeline.spec.phase import TimeDomainPhaseCorrectionTask
sdt=timeDomainPhaseCorrection(sdt=sdt, nhkt=nhkt\
                              lpfPar = obs.calibration.spec.lpfPar, \
                              phaseCorrLim = obs.calibration.spec.phaseCorrLim, \
                              chanTimeConst = obs.calibration.spec.chanTimeConst)
```

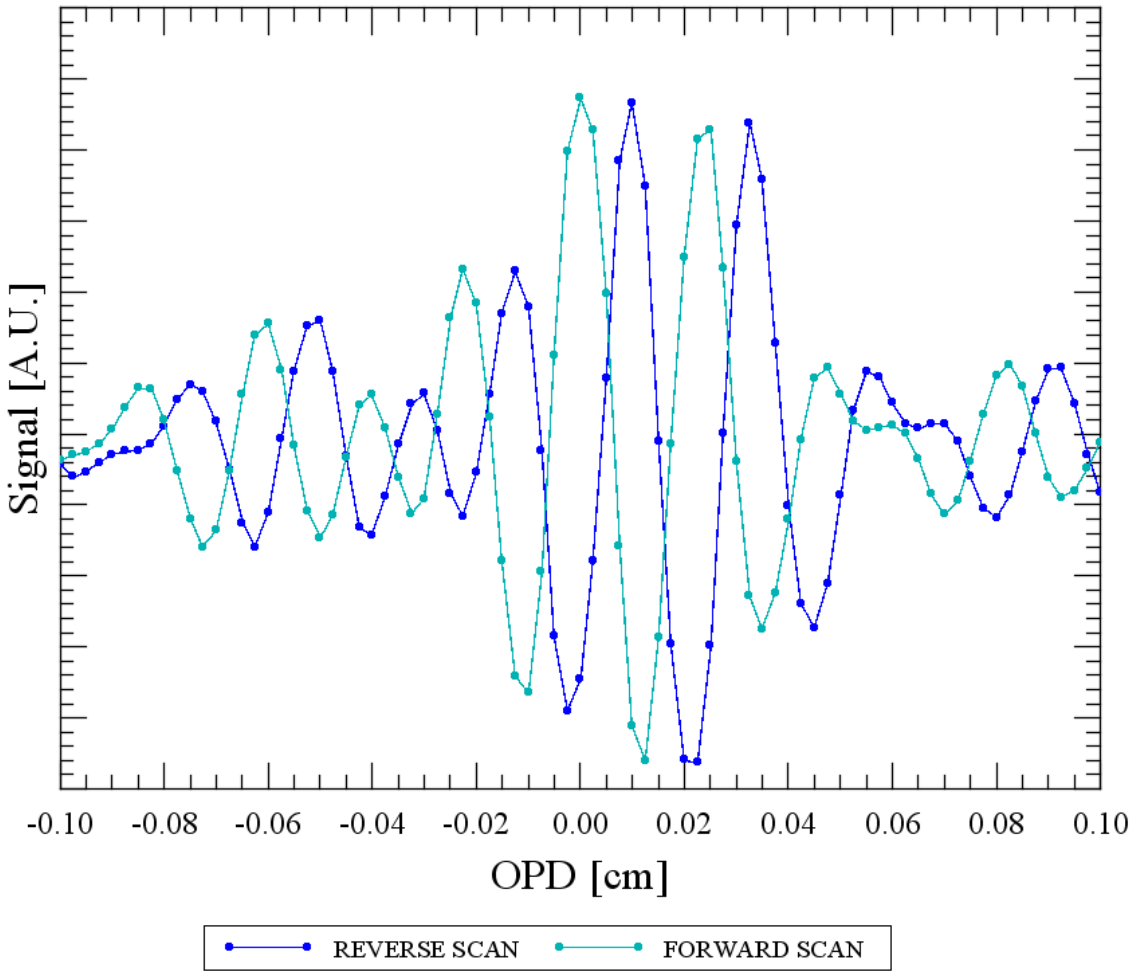

Figure 6.4. Forward and reverse interferograms do not line up well without applying time domain phase correction. A forward (green) and a reverse (blue) scan are slightly shifted with respect to one another.

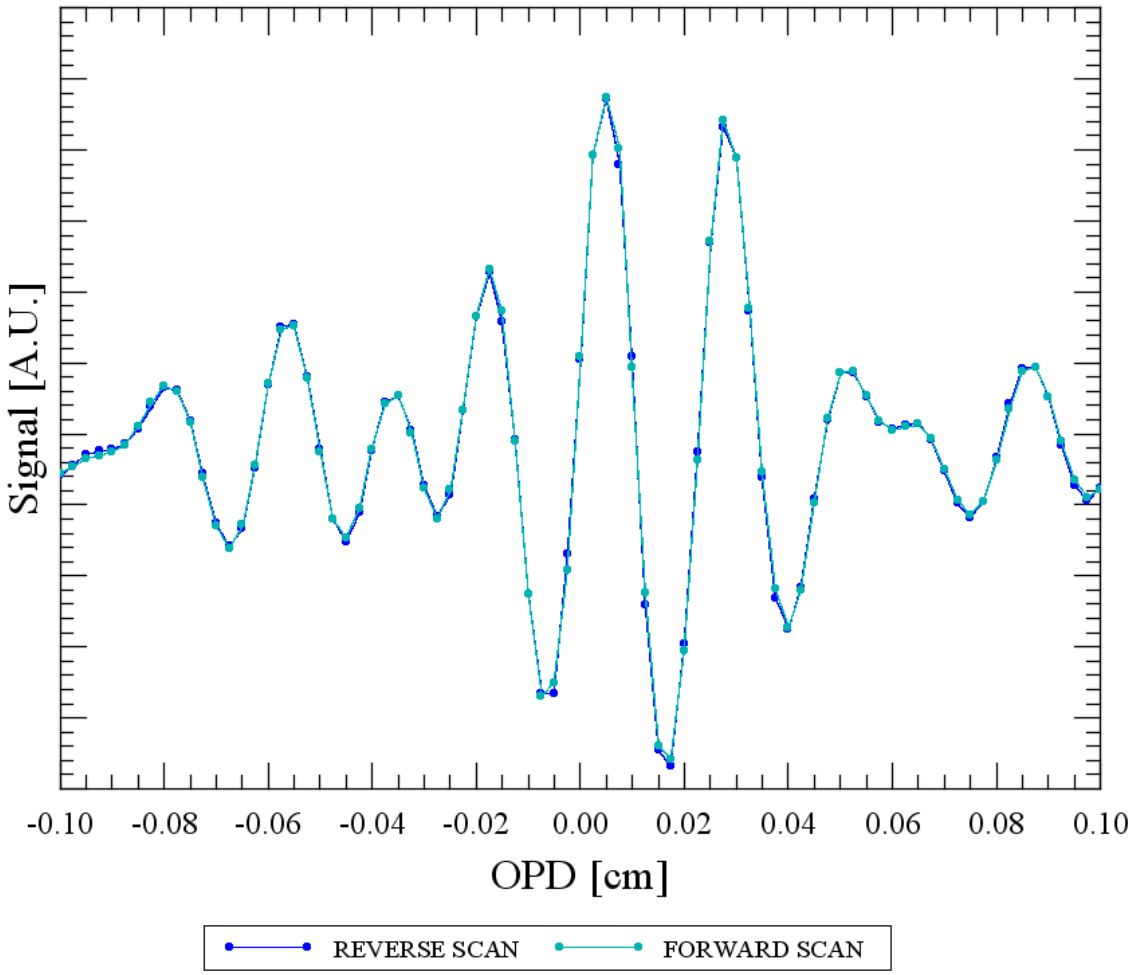

Figure 6.5. Forward and reverse interferograms line up much better after applying time domain phase correction to the Spectrometer Detector Timelines. Forward (green) and reverse (blue) scan.

6.7.8. Error messages

There are no specific error messages defined for this task.

6.8. Conversion of the BSM telemetry into Angles on the Sky

6.8.1. Module Description

The *Calculate BSM Angles* Module converts the angles contained in the Beam Steering Mirror Timeline (BSMT) to physical units (decimal degrees on the sky). The method is simple, interpolating the values in a calibration table, which relates the two raw signal values from the BSMT (in the BSM chop and jiggle axes) to the two converted angles (Y and Z axes which are relative to the spacecraft). Alternatively, the Nominal Housekeeping Timeline (NHKT) can be provided to the task instead of the BSMT. The NHKT also contains the BSM sensor values, however sampled at 1Hz rather than 16Hz.

6.8.2. Input Data Products

One data product of type `EdpProduct` is required as input. This product must contain the timeline of the two angles of the mirror (chop and jiggle) in raw data, either a Beam Steering Mirror Timeline or a Nominal Housekeeping Timeline.

BSMT — **Beam Steering Mirror Timeline** The input product contains the time samples and corresponding BSM sensor signal values in raw format in the `chopSens` and `jiggSens` parameters for each observation building block, sampled at a frequency of 16Hz.

NHKT — **Nominal Housekeeping Timeline** The input product contains the time samples and corresponding BSM sensor signal values in the NHKT in raw format in the `chopsenssig` and `jiggsenssig` parameters for each observation building block, sampled at a frequency of 1Hz.

6.8.3. Output Data Products

BAT — **BSM Angles Timeline** The output BAT product contains timelines for the angular offset (in spacecraft Y,Z angles) of the BSM with respect to its rest position (defined by the minimum power dissipation of the BSM mechanism) on the sky in units of arc seconds for each observation building block.

6.8.4. Input Calibration Products

BSM Position Table — **BSM Position Table** This Table contains the BSM sensor signals in raw units (in chop and jiggle directions) and angular distance on the sky from its zero position in arcsecs (in spacecraft Y, Z coordinates). There is one table for the photometer and one for the spectrometer, because the reference BSM position can be different.

6.8.5. How to use the *Calculate BSM Angles* Module

6.8.6. Parameter Options

The Calculate BSM Angles module does not use any parameters that can be controlled by the user during run-time. The interpolation method is not a parameter.

6.8.7. Examples

```
from herschel.spire.ia.pipeline.common.bsm import CalcBsmAnglesTask
bat = calcBsmAngles(bsmt, bsmPos=obs.calibration.spec.bsmPos)
bat = calcBsmAngles(nhkt, bsmPos=obs.calibration.spec.bsmPos)
```

The process is represented pictorially in the figure below.

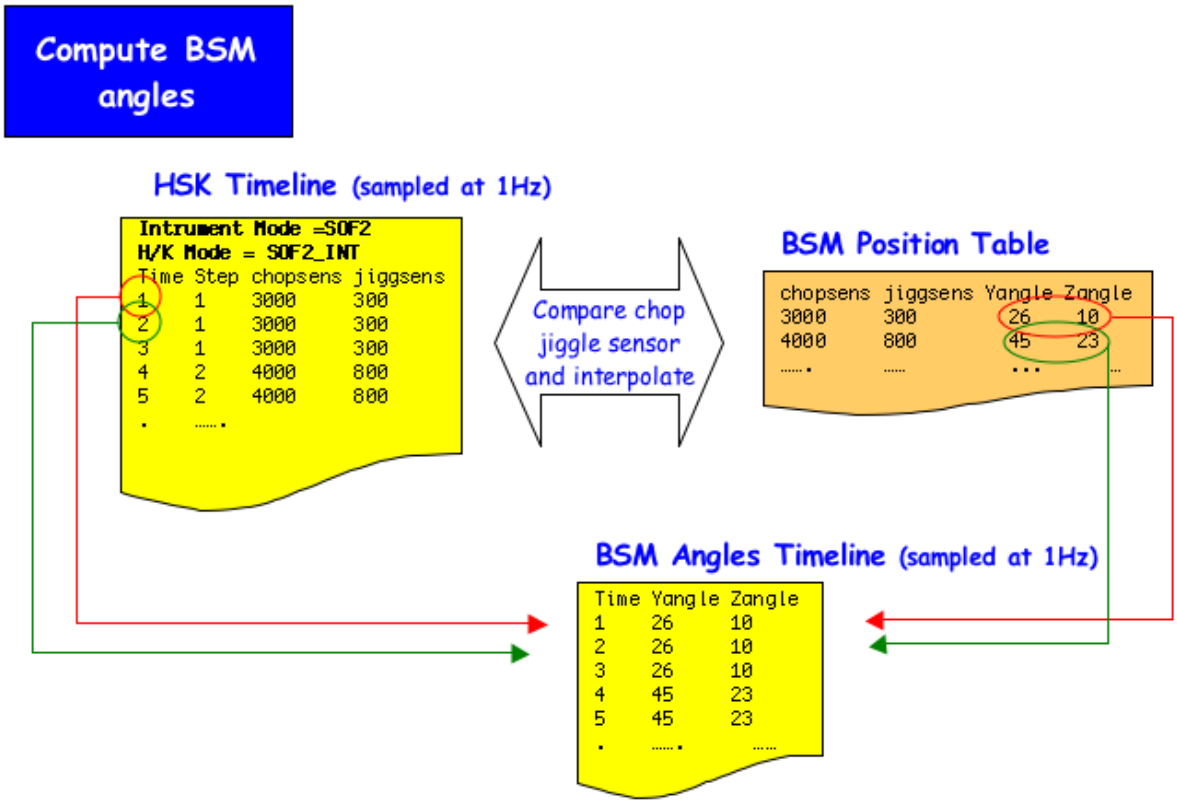

Figure 6.6. Creation of BSM Angles Timeline by the Calculate BSM Angles module in the Spectrometer Pipeline.

6.8.8. Error messages

error: herschel.ia.task.SignatureException: Error in calcBsmAngles Task:the input product must be either a Beam Steering Mirror Timeline (BSMT) or a Nominal Housekeeping Timeline (NHKT) The input product type was checked and found not to be a BSMT or a NHKT. The module can only operate on these two product types.

6.9. Create the SPIRE Pointing Product

6.9.1. Module Description

This class creates a SPIRE Pointing Product.

6.9.2. Input Observational Data Products

HPP — **Herschel Pointing Product** The HPP contains the sky position along the line of sight of the Herschel telescope for the full Operational Day.

SIAM — **Spacecraft Instrument Alignment Matrix** The SIAM product contains the relative angles between the Herschel telescope line of sight and the invidiual apertures of the detector arrays of the SPIRE spectrometer.

BAT (Optional) **Beam Steering Mirror Angles Timeline** The Beam Steering Mirror (BSM) Angles timeline specifies the angular offset along the two spacecraft axes Y and Z for a given time due to the position of the BSM.

6.9.3. Output Data Products

SPP **SPIRE Pointing Product** The SPIRE Pointing Product is a context containing the Herschel Pointing Product, the Spacecraft Instrument Alignment Matrix, the Detector Angular Offset calibration product and the BSM Angles Timeline. This class allows the user to obtain the position of one or more SPIRE detectors on the sky at a given time.

6.9.4. Input Calibration Products

detAngOff **Detector Angles Offset** This calibration product contains the relative angles between the center detectors and the off-center detectors of the two detector arrays of the SPIRE spectrometer.

6.9.5. How to use the *Create SPIRE Pointing Product* Module

The SpirePointingProduct calculates the pointing for each detector by adding the value stored in the DetAngOff table (gives the offset of each detector on the sky from the instrument boresight) and adding the value from the BSM Angles Timeline to the pointing of the central aperture, which is derived from the BSM Positions calibration from the measured CHOP and JIGG sensor values.

The BSM Positions calibration is calculated by observing what CHOPSENS and JIGGSENS place the source (i.e. the instrument boresight, or HPP+SIAM) on a particular detector, it gives the angle on the sky through which the instrument boresight has moved on the sky. In order to calculate the final pointing at each detector in the array, the different contributions to the pointing would need to be combined as follows:

HPP + SIAM + DetAngOff - BAT

where the BAT is subtracted so that it gives the angle that the boresight has moved on the sky, which is in the opposite sense to the correction that must be applied to the actual sky coordinates for each detector.

6.9.6. Parameter options

The create SPP module does not use any parameters that can be controlled by the user at run-time.

6.9.7. Examples

Assuming `bat` is a valid BSM Angles Timeline product and `obs` an observation context, the SPP can be created as follows:

```
from herschel.spire.ia.pipeline.common.pointing import CreateSpirePointingTask
spp = createSpirePointing(hpp=obs.auxiliary.pointing, \
                          siam=obs.auxiliary.siam, \
                          detAngOff=obs.calibration.spec.detAngOff, \
                          bat=bat)
```

Assuming `sdt` is a valid Spectrometer Detector Timeline Product, one can obtain the sky position of the SSWA1 and SSWD4 apertures from the SPP at the beginning of this timeline, using information from the spacecraft gyroscopes:

```
time=sdt.startDate
A1,D4=spp.getSkyPosition(["SSWA1","SSWD4"],time,"gyro")
```

6.9.8. Error messages

There are no specific error messages defined for this task.

6.10. Creating the Interferograms from the Timeline Data

6.10.1. Module Description

A single building block of a SPIRE spectrometer observation in scanning mode consists of a series of scans of the spectrometer mechanism (SMEC) while the instrument is pointed at a given target. The sampling of the SPIRE spectrometer detectors and the spectrometer mechanism is not synchronized; the two subsystems are sampled at different rates and at different times. In order to derive the source spectrum from the measured data, the spectrometer detector samples must be linked with the position of the SMEC in the form of interferograms, i.e. signal as a function of optical path difference (OPD). Additionally, the spectrometer detector signal timelines are interpolated onto timelines where the SMEC positions are equidistantly spaced to ensure accurate transformation of the interferogram with the Discrete Fourier Transform. First, the SMEC timeline is interpolated from one that is non-equidistant in position to one that is equidistant in position. Then, the detector signal timelines are interpolated onto the times which correspond to the equidistant SMEC position grid. The mean value of the sky position during a given scan is assigned to the interferograms in the output SDI product.

6.10.2. Input Data Products

SDT	**Spectrometer Detector Timeline** The SDT contains the measured signal sample timelines for each spectrometer detector in units of Volts. The units are expected to be identical for all channels.
SMECT	**Spectrometer Mechanism Timeline** This product contains the timelines for the telemetry parameters related to the spectrometer mechanism (SMEC). The unit of the coarse and fine optical encoder readings is expected to be centimetre.
SPP (Optional)	**SPIRE Pointing Product** The SPIRE Pointing Product is a context containing the Herschel Pointing Product, the Spacecraft Instrument Alignment Matrix, the Detector Angular Offset calibration product and the BSM Angle Timeline. This class allows the user to obtain the position of one or more SPIRE detectors on the sky at a given time.
NHKT	**Nominal Housekeeping Timeline** This product contains the timelines for the nominal housekeeping telemetry parameters. The housekeeping parameters used by this module are: • **SCANSTART** This quantity contains the commanded start position for the SMEC for the observation in centimetres. All entries in this timeline should be identical. • **SCANEND** This quantity contains the commanded end position for the SMEC for the observation in centimetres. All entries in this timeline should be identical.

- **SCANS** This quantity contains the value of the scan number. This quantity is expected to be unitless and its values should only decrement over the course of the observation.

6.10.3. Output Data Products

SDI	**Spectrometer Detector Interferogram** The SDI is an output product that contains the interferograms for each spectrometer detector for each scan of the observation. The signal quantities of this output product will be in units of Volts - identical to the signals of the input SDT product. The quantities of the OPD columns of this product will be in units of centimetres.

6.10.4. Input Calibration Products

Calibration Context (Optional)	**Calibration Context** Contains all calibration products. Individual products can be passed in to overwrite those in the context.
smecZpd (Optional)	**Zero Path Difference** This calibration product contains the values of the position of zero path difference (ZPD) for each SPIRE spectrometer detector. There are two entries for each detector; one entry gives the SMEC optical encoder value for ZPD in units of centimetres, the other entry gives the LVDT value for ZPD.
chanTimeOff (Optional)	**Spectrometer Channel Time Offset** This calibration product contains the time offsets between the frametime and the readout time for the spectrometer detectors in the SDT product. The quantities in this calibration product are expected to be in units of seconds.
smecStepFactor (Optional)	**Optical Encoder to Optical Path Difference** This calibration product contains the conversion factors that relate a step reported by the SMEC encoder to a step in OPD. Nominally, this conversion factor is equal to 4 for a Mach-Zehnder FTS. For the SPIRE FTS this conversion factor will be slightly different for each detector and depend on the off-axis angle. Each entry is expected to be unitless.

6.10.5. How to use the *Interferogram Creation* Module

6.10.6. Parameter Options

Users can set the following parameters of the Interferogram Creation module:

interpolType (Optional)	This parameter specifies the type of interpolation to be performed. Currently, this parameter can be set to one value only: the default value is "spline", indicating that cubic spline interpolation will be used. This simple interpolation method has the advantage of being robust and has been found not to degrade the quality of the data from the SPIRE FTS.

6.10.7. Examples

Assuming `sdt` to be a detector timeline of SPIRE spectrometer data processed to Level 0.5, with bolometer signal data in units of Volts, `smect` to be a SMEC timeline, `spp` to be a SPIRE pointing product and `obs` to be an observation context:

```
from herschel.spire.ia.pipeline.spec.ifgm import CreateIfgmTask
sdi=createIfgm(sdt=sdt, smect=smect, nhkt=nhkt, spp=spp, \
          cal=obs.calibration, interpolType="spline")
```

The calibration context can be passed in as a single parameter for convenience. However, only the calibration products smecZpd, chanTimeOff, and smecStepFactor are used. These calibration input products can also be passed individually to override the ones given in the context:

```
sdi=createIfgm(sdt=sdt, smect=smect, nhkt=nhkt, spp=spp, \
          cal=obs.calibration, smecZpd=mySmecZpd, \
          interpolType="spline")
```

6.10.8. Error messages

error: SCANSTART/SCANEND not repeating. This module will throw an exception if the value of the housekeeping parameters SCANSTART/SCANEND change. These parameters should specify the commanded scan start and end positions and should therefore not change during a building block.

error: All scans have dMPD less than or equal to 0.

error: SDI is empty

6.11. Removing the Telescope and Calibration Source Background from the Data

6.11.1. Module Description

The SPIRE FTS measures a signal from the astronomical source but also from the warm Herschel telescope and the Spectrometer Calibration Source (SCal). The *SCAL and Telescope Correction* Module aims to eliminate the contribution from the telescope and SCal by subtracting a deep reference measurement of a dark region of the sky.

6.11.2. Input Data Products

SDI	**Spectrometer Detector Interferogram** This product contains interferograms for each spectrometer detector for each scan of the observation building block.
NHKT (Optional)	**Nominal HouseKeeping Product** This product contains temperature timelines for SCal.

6.11.3. Output Data Products

SDI	**Spectrometer Detector Interferogram** The output SDI product contains the corrected interferograms for each spectrometer detector for each scan of the observation building block.

6.11.4. Input Calibration Products

interRef	The SpecInterRef Calibration product contains the interferogram from a deep observation of a dark region in the sky for each spectrometer detector and the two directions (forward and reverse) of the linear translation stage (SMEC) of the SPIRE FTS. There are different editions depending on the selected spectral resolution (LR, MR, HR), the bias mode of the detectors (nominal or bright), and the time of the observation.

6.11.5. How to use the *SCAL and Telescope Correction* Module

6.11.6. Parameter Options

The Telescope and SCal Correction module does not use any parameters that can be controlled by the user at run-time.

6.11.7. Examples

Assuming an `sdi` spectrometer detector interferogram product and `obs` is an observation context:

```
from herschel.spire.ia.pipeline.spec.scal import TelescopeScalSubtractionTask
# get the commanded resolution from the SDI ("HR", -"MR" or -"LR")
commandedResolution = sdi.meta['commandedResolution'].value
# get the bias mode (biasMode) from the SDI ("nominal" or -"bright")
biasMode = sdi.meta["biasMode"].value
# get the observation start date from the SDI
date = sdi.startDate
# get the correct edition of the calibration product
interRef = obs.calibration.spec.interRefList.getProduct(commandedResolution, \
            biasMode, date)

# execute the task providing it the correct interRef calibration product
sdi = telescopeScalSubtraction(sdi, nhkt=nhkt, interRef=interRef)
```

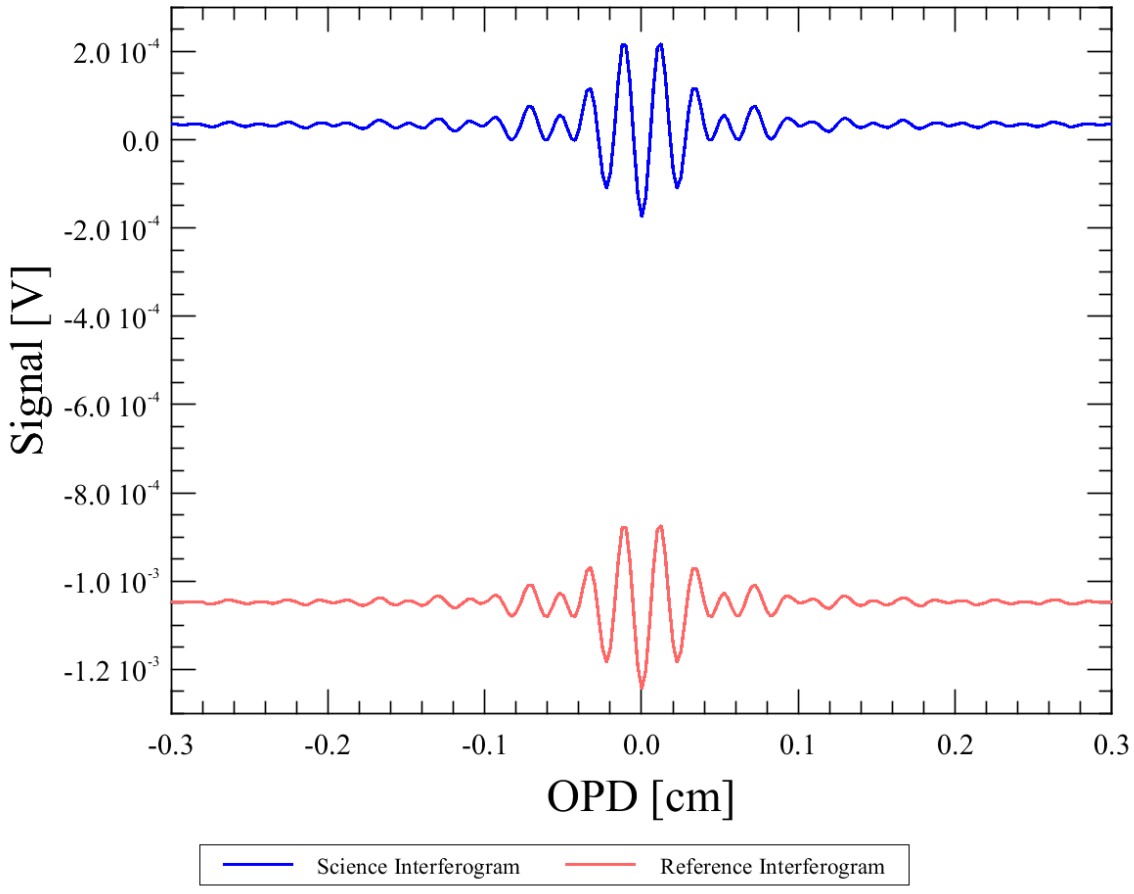

Figure 6.7. Interferogram from a scientific observation and a dark sky reference.

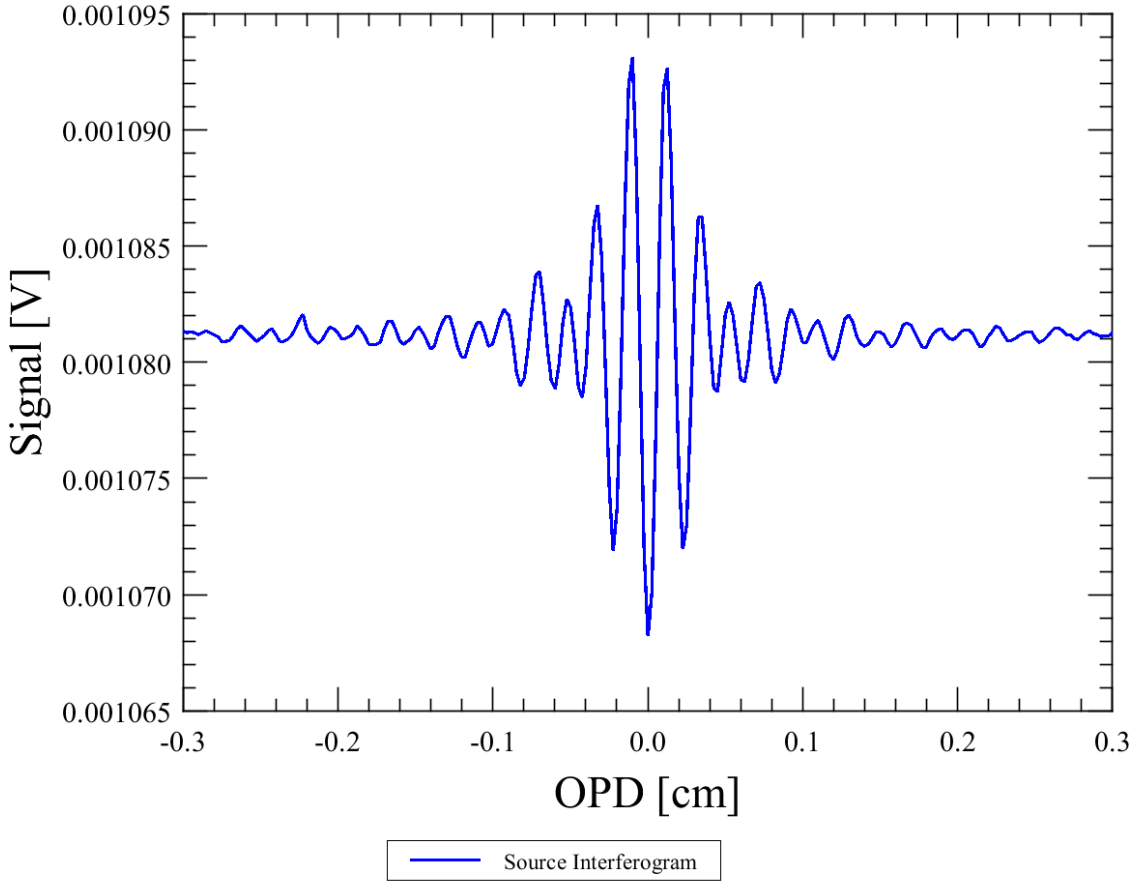

Figure 6.8. Interferogram from the source.

6.11.8. Error messages

error: severe: Calibration product SpecInterRef is not compatible with Enter product return = sdi enter. The module issues an exception if the input SDI could not be corrected due to a nonsuitable calibration product.

6.12. Removing the Baseline from the Interferograms

6.12.1. Module Description

The intensity incident on the SPIRE spectrometer detectors can be separated into two components: a component that is constant as a function of OPD and a component that is modulated as a function of OPD. As the first baseline term does not contain relevant spectral information, it may be removed without affecting the source spectrum. Frequency components outside of the optical passband can also be removed from the second term. On a detector-by-detector and scan-by-scan basis, the baseline correction algorithm evaluates and removes the baseline of the interferogram. The task offers two options to characterize the baseline: Either a polynomial fit or the inverse Fourier transform of the spectrum including only low frequencies.

6.12.2. Input Data Products

SDI — **Spectrometer Detector Interferogram** The input SDI product contains interferograms for each spectrometer detector for each scan of the observational building block.

6.12.3. Output Data Products

SDI — **Spectrometer Detector Interferogram** The output SDI product contains the corrected interferograms for each spectrometer detector for each scan of the observational building block.

6.12.4. Calibration Data Products

No calibration data are needed.

6.12.5. How to use the *Baseline Correction* Module

6.12.6. Parameter Options

Users can set the following parameters of the Baseline Correction module:

type (Optional) — This parameter determines the type of baseline fitting to perform. The user has two options: "polynomial" or "fourier". Set this parameter to "polynomial" to perform a least-square minimized polynomial fit (default option) or to "fourier" to fit the baseline with the low frequency Fourier components of the spectrum that corresponds to the interferogram. Both options are of comparable quality in the sense of removing any baseline variation in the interferograms and not significantly affecting the spectral content in the optical passbands, i. e. spectral artifacts are small compared to instrumental noise. Polynomial fitting is a good option if the shape of the interferogram baseline is dominated by optical vignetting. Fourier fitting is the preferred option when 1/f-like noise dominates the shape of the interferogram baseline. 1/f-like noise has been observed during the Performance Verification of the SPIRE spectrometer when observing bright, point-like sources.

Note that both algorithms will poorly fit to the baselines of interferograms which are truncated and have not been repaired by the clipping correction module.

degree (Optional) — If the fitting type is "polynomial" then "degree" specifies the order of the polynomial function to fit to the baseline of the input interferograms. The default value is 4. The degree of the polynomial function should be even because of the even symmetry inherent to the FTS. An order of 2 approximates the baseline slightly worse and introduces stronger spectral artefacts in the optical passbands. Polynomial functions with orders larger than 4 are possible but do not usually improve the results significantly.

threshold (Optional) — If the fitting type is "fourier" then "threshold" specifies the maximum frequency in inverse centimeters to include when using the Fourier components to fit to the baseline. 4 cm^{-1} has been found to be a good value for the specific purposes of the SPIRE FTS in terms of achieving a good fit to the baseline and introducing spectral artefacts below the noise. The threshold should always be lower than the lower edges of the optical passbands, i.e. 15 cm^{-1} for SLW and 30 cm^{-1} for SSW.

If interferograms display unusual artefacts such as temporary deviations from the baseline (sudden steps, pepper and salt noise) then a Fourier type fitting with a high threshold (e.g. 10 cm^{-1}) may be effective at removing such unwanted features.

6.12.7. Examples

```
from herschel.spire.ia.pipeline.spec.baseline import BaselineCorrectionTask
sdi=baselineCorrection(sdi=sdi, type="polynomial", threshold=4, degree=4)
```

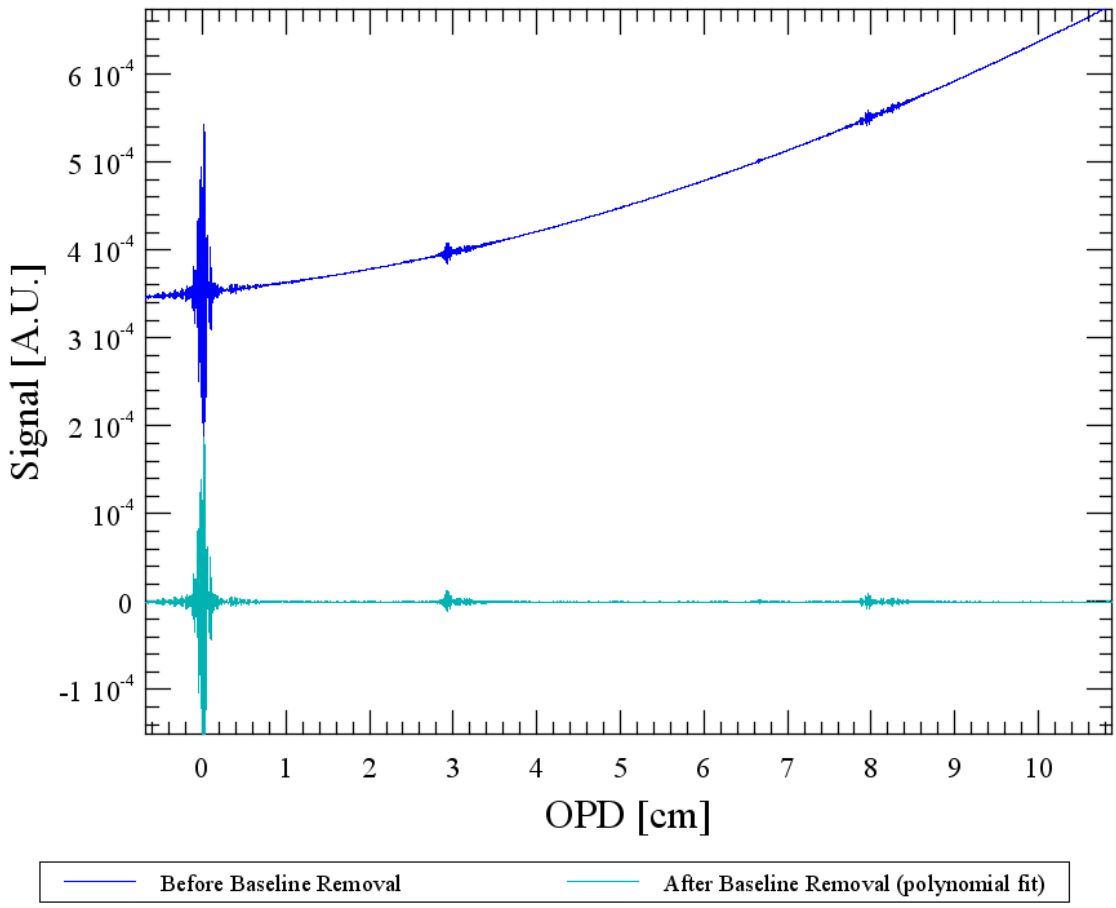

Figure 6.9. An example for removing the baseline with a 4th order polynomial fit to the interferogram. The interferogram before (blue) and after (green) baseline removal.

6.12.8. Error messages

error: Invalid baseline correction type specified: "xx" The type control parameter must be either omitted and set to one of the following two strings: "polynomial" (default) or "fourier".

error: Output SDI is empty.

error: OPD units are "xx". Expected units of cm. The x-axis of the interferograms is OPD. The task assumes that values are given in centimeters if no units are specified for the OPD. If units are indeed specified for the OPD, then it must be in units of centimeters. Otherwise the module will throw an exception.

6.13. Removing Glitches from the Interferograms

6.13.1. Module Description

The *Interferogram Deglitching* Module identifies and removes glitches in the interferograms for each spectrometer detector. On an OPD-position-by-OPD-position basis, the samples from one scan are compared to those from all other scans in the same observation. Two different methods are available to compare the data and flag outliers. The baseline approach is to use the median of the spectral data across scans and a threshold factor times the standard deviation or the median absolute deviation to define a range for outliers. The alternative approach uses a window of a user-defined length within which the standard deviation is computed and outliers are flagged based on a standard deviation or median absolute deviation. The scan with the largest absolute deviation will then be discarded and the procedure repeated until no further outliers are flagged. The flagged samples are then replaced by the average of the non-glitch samples from the other observed interferograms at that position.

6.13.2. Input Data Products

SDI — **Spectrometer Detector Interferogram** The input SDI product contains interferograms for each spectrometer detector for each scan of the observational building block.

6.13.3. Output Data Products

SDI — **Spectrometer Detector Interferogram** The output SDI product contains interferograms for each spectrometer detector for each scan of the observational building block which have been corrected for glitches.

6.13.4. Input Calibration Products

No calibration data are needed.

6.13.5. How to use the *Interferogram Deglitching* Module

6.13.6. Parameter Options

deglitchType (Optional) — This optional parameter specifies the method used to identify glitches. Values may be either "STD", "MAD", "STD_WINDOW", or "MAD_WINDOW". If "STD" or "MAD" are selected then glitches are identified by a 1st order outlier criterion based on the difference of a given sample from the median value across all scans at a given Optical Path Difference (OPD) in units of either standard deviation or median absolute deviation. If the deviation is greater than a user-defined threshold (see keyword below) then the sample is flagged as a glitch. This commonly used approach to outlier detection is suitable for large iteration numbers. If "STD_WINDOW" or "MAD_WINDOW" are selected then glitches are identified by a 2nd order outlier criterion based on the standard deviation interferogram across all scans. If, within the user-defined window (see keyword below), the standard deviation interferogram contains outliers above a threshold factor (see keyword below) times either standard deviation

	or median absolute deviation, then that OPD location is identified as containing a glitch. The scan with the largest deviation from the median is then flagged as a glitch and the procedure is iterated until no more glitches are found or only two scans are left. All of the deglitching algorithms require a total of at least three valid scans. The algorithms ignore samples flagged as unusable. By default, the module will use the "MAD_WINDOW" glitch detection algorithm with its default parameters.
thresholdFactor (Optional)	This parameter specifies the factor by which the standard deviation or the median absolute deviation is multiplied to define the range of the envelope outside of which samples are flagged as outliers. This parameter applies to all four types of deglitching algorithms, if in slightly different ways for 1st and 2nd order outlier detection. A larger threshold will result in less sensitive glitch detection and fewer data points which are incorrectly flagged as glitches (false positives or ghosts). A typical threshold factor for large sample sizes is a value of 3 when using standard deviation. Note that the module will multiply the threshold factor by a factor of 1.4826 for algorithms using the median absolute deviation in order to make the results comparable with algorithms using the standard deviation. The module sets default threshold values based on the number of scans in the observation if the user does not specify the threshold value. The default value is 3.2905 and 4.8785 for "STD" and "MAD", setting the expectation for false positives in white noise to 0.1%. The default values for the other two deglitching schemes depend on the number of scans in a given direction. The default values for "STD_WINDOW" are 4.0, 3.5, 3.0, and 2.5 for 2, 3, 4 - 8, and >9 scans in one direction. The default values for "MAD_WINDOW" are 4.5, 4.0, 3.5, and 3.0 for 2, 3 - 4, 5 - 8, and >9 scans in one direction. If, in either direction, the number of scans is one or two, the scans from the other direction will be used in addition to enable the statistical analysis. The module cannot identify and correct for glitches if the total number of scans is less than three.
windowSize (Optional)	This parameter of type integer specifies the window size for the deglitching types "STD_WINDOW" and "MAD_WINDOW". It must be an odd number in order to define a symmetrical data range around the sample to be analyzed. Its value must be equal to three or larger. The default values for the deglitching types "STD_WINDOW" and "MAD_WINDOW" are 41 and 33 respectively. It is recommended to keep this parameter smaller than the width of the center burst of the measured interferograms. This parameter has no bearing on the two deglitching types "STD" and "MAD".
getExpectedFalsePositives-Ratio (deglitchType, nScans, nElements, threshold, windowSize)	Deglitching interferograms allows for a large number of different parameter combinations and it is not necessarily obvious which combination to select. The task provides a method to assess the performance of a specific parameter set for the data at hand in terms of the number of false positives. The getExpectedFalsePositivesRatio method returns the number of samples which were wrongly flagged as glitches in one set of normally distributed data divided by the total number of samples. In the order indicated above, the method requires five parameters: deglitchType, i.e. "STD_WINDOW", "MAD_WINDOW", "STD", or "MAD". nScans, the number of scans along one direction.

	nElements, the number of elements in one interferogram (to account for edge effects). threshold, the threshold factor. windowSize, the number of elements in the window if deglitchType is "STD_WINDOW" or "MAD_WINDOW". This parameter must be set even if deglitchType is "STD" or "MAD" where it has no effect. An example of how to use this function is given below.
identifyGlitches (Optional)	This boolean parameter with default value True specifies whether the task attempts to identify glitches. This option was made available to allow for purely manual glitch detection.
correctGlitches (Optional)	This boolean parameter with default value True specifies whether the task attempts to replace flagged samples.

6.13.7. Examples

```
from herschel.spire.ia.pipeline.spec.ifgm.deglitch import DeglitchIfgmTask
sdi=deglitchIfgm(sdi=sdi, \
                    deglitchType = -"MAD_WINDOW", \
                    windowSize = 33, \
                    thresholdFactor = 5.0)
print -"In randomly distributed data, the following number of false positives
       were flagged when applying this algorithm to 6 interferograms along
       the same direction with 2000 samples each:"
for i in range(10):
   print 6*2000*deglitchIfgm.getExpectedFalsePositivesRatio("MAD_WINDOW", 6,
2000, 5.0, 33)
```

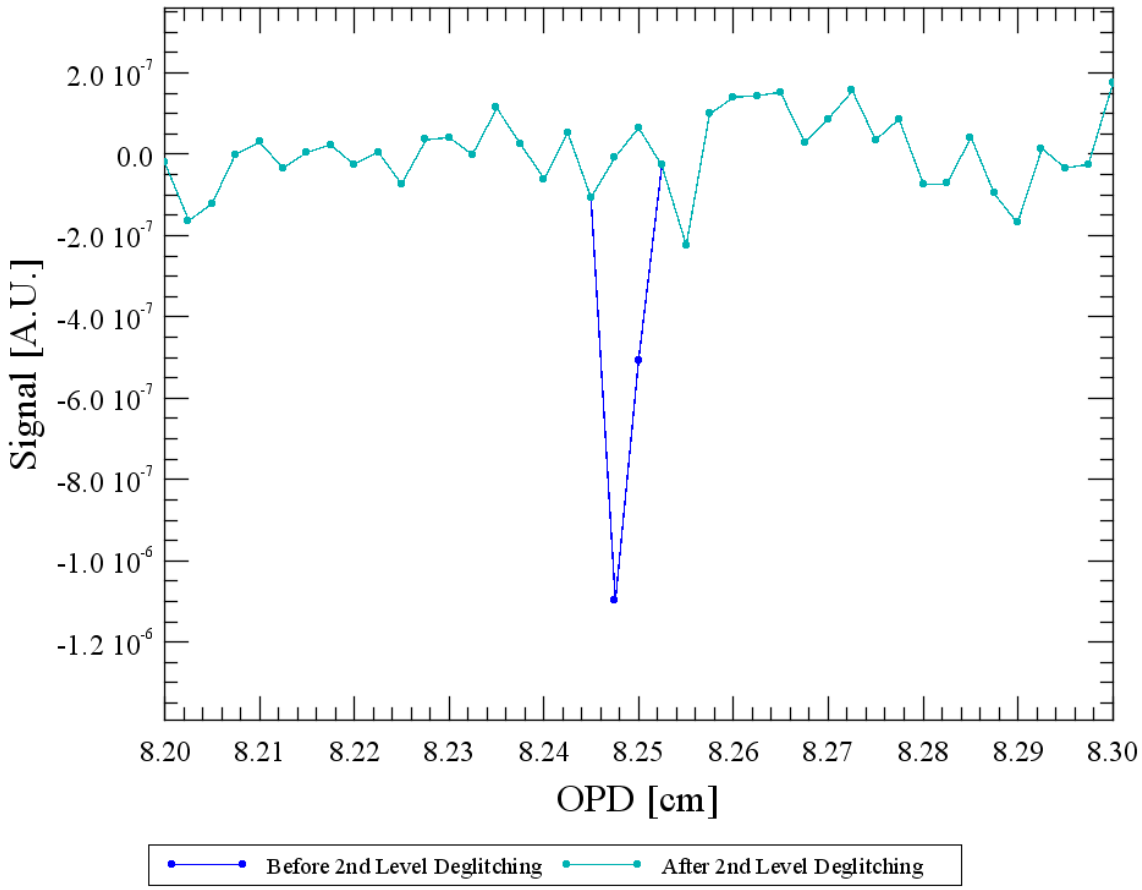

Figure 6.10. An example for glitch removal in an interferogram. An interferogram before (blue) and after glitch removal (green).

6.13.8. Error messages

There are no specific error messages defined for this task.

6.14. Applying an Apodization Function to an Interferogram

6.14.1. Module Description

The *Apodization* module multiplies the interferograms for each detector and scan of an observation building block with an analytically defined tapering function. The purpose of this processing step is to reduce the sidelobes of the instrumental line shape of the SPIRE spectrometer.

6.14.2. Input Data Products

SDI — **Spectrometer Detector Interferogram** This product contains interferograms for each spectrometer detector for each scan of the observational building block.

6.14.3. Output Data Products

SDI — **Spectrometer Detector Interferogram** This product contains apodized interferograms for each spectrometer detector for each scan of the observational building block. If pre-apodization is selected then the output SDI product will contain double-sided interferograms. If post-apodization is selected then the output SDI product will contain samples for only those OPDs where OPD >= ZPD.

6.14.4. Input Calibration Products

No calibration data are needed.

6.14.5. How to use the *Apodization* Module

6.14.6. Parameter Options

apodType — **Apodization Type** This mandatory parameter defines the type of apodization that is to be performed - either double-sided or single-sided. The distinction is determined by whether the task is run before or after the phase correction step. Therefore, the possible values of "apodType" are "prePhaseCorr" and "postPhaseCorr".

In the "prePhaseCorr" mode, a double-sided apodization is used. The chosen apodization function is applied only to the symmetric portion of the incoming interferograms (OPD = {-N/2 ..., 0, ..., N/2}). Those samples of the incoming interferograms that fall outside the aforementioned range are removed, and a new SDI product containing these symmetric, apodized interferograms is returned to the user (i.e. the original SDI is not changed).

In the "postPhaseCorr" mode, the apodization function is applied only to the single-sided portion of the interferograms, i.e. the selected apodization function is applied only to those samples whoe OPD is greater than or equal to zero (OPD = {0, ..., N/2}). Those samples whose OPD are less than zero are removed from the interferograms, and a new SDI product containing these symmetric, apodized interferograms is returned to the user (i.e. the original SDI is not changed).

apodName (Optional) — **Apodization Function Name** This optional parameter defines the apodizing function that is to be applied to interferograms in the input SDI product. By default, the apodization task uses the apodizing function aNB_15.

In addition to the long-standing apodizing functions Hamming, Hanning, and Gaussian, the SPIRE data processing environment offers ten functions that optimize the tradeoff between minimizing the secondary sidelobes of the instrumental line shape and reducing spectral resolution. The name of each apodizing function indicates the loss of spectral resolution. The factors run from 1.1 to 2.0 in steps of 0.1, indicating that the full width at half maximum of an unresolved line feature will increase by this factor. The names of the apodizing functions available for this task are as follows: aNB_11 ... aNB_20, where aNB stands for Norton-Beer. These apodizing functions are optimal in the sense of minimizing the secondary sidelobes of the instrumental line shape for the specified reduction in spectral resolution. Further details can be found in Naylor, D.A., and Tahic, M.K., "Apodiz-

ing functions for Fourier transform spectroscopy". J. Opt. Soc. Am. A 24, 3644-3648 (2007).

The following apodizing functions are available:

- "aGauss_19", Gaussian
- "aHM_15", Hamming
- "aHN_17", Hanning
- "aNB_11" - "aNB_20", ten adjusted Norton Beer functions (aNB_15 is default)

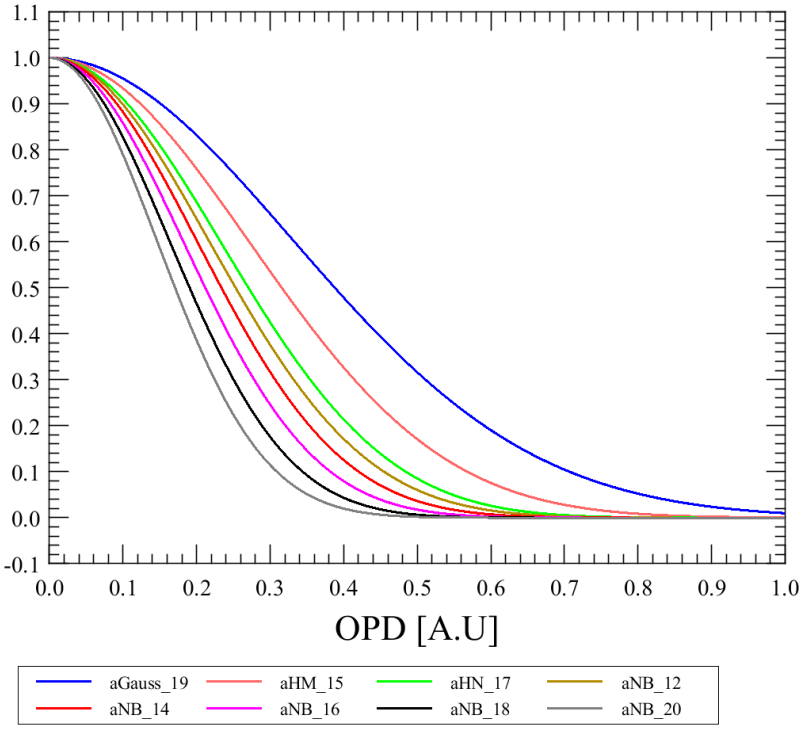

Figure 6.11. A selection of the available apodizing functions. The x-axis is OPD in arbitrary units. The apodizing functions will always be stretched to the maximum OPD. Some of the adjusted Norton-Beer functions are omitted for clarity.

6.14.7. Examples

The following is an example of pre-apodization, i.e. apodizing double-sided interferograms in the SDI prior to phase correction.

```
from herschel.spire.ia.pipeline.spec.apodize import ApodizeIfgmTask
presdi=apodizeIfgm(sdi, apodType="prePhaseCorr", apodName="aNB_15")
```

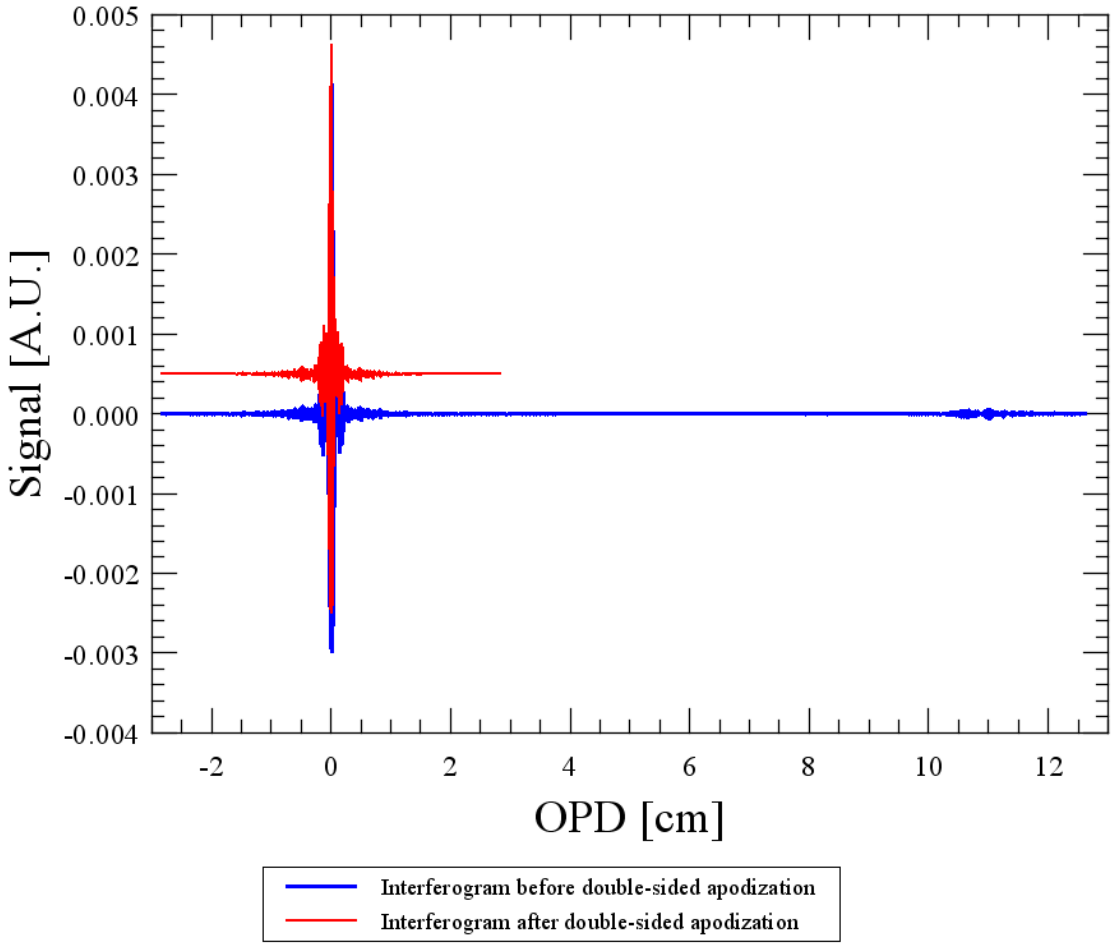

Figure 6.12. An example for double-sided apodization: The original interferogram (blue) and the apodized interferogram (red), offset by 0.0005 for clarity. The apodizing function aNB_15 was used.

The following is an example of post-apodization, i.e. apodizing single-sided interferograms in the SDI after phase correction.

```
sdi=apodizeIfgm(sdi, apodType="postPhaseCorr", apodName="aNB_15")
```

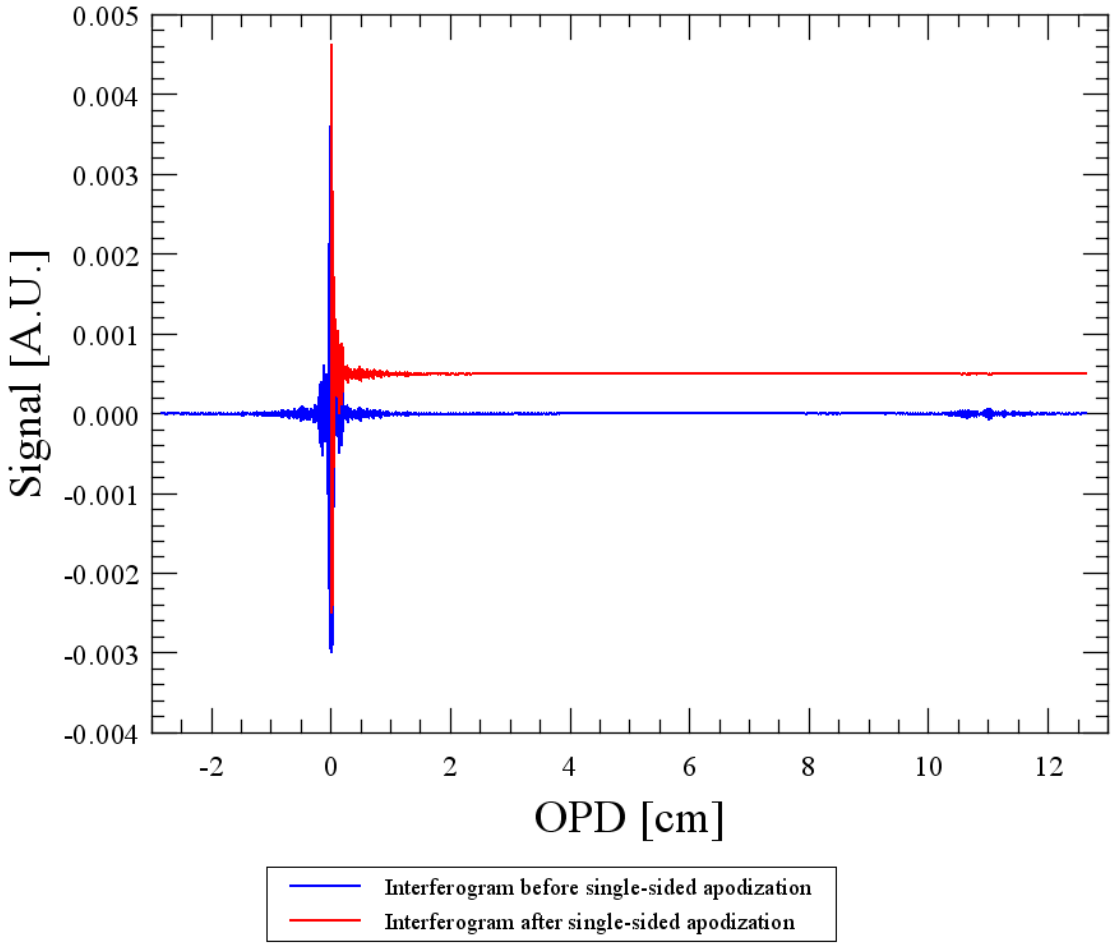

Figure 6.13. An example for single-sided apodization: The plot shows the original interferogram (blue) and the apodized interferogram (red), offset by 0.0005 for clarity. Note how the channel fringe feature at high OPD is deprecated in the apodized interferogram. The apodizing function aNB_15 was applied.

6.14.8. Error messages

error: Cannot get double sided portion: Range of OPD grid does not contain 0. This module will throw an exception if the interferogram does not contain ZPD.

error: Cannot get double sided portion: ZPD cannot be determined. This module will throw an exception if the interferogram does not contain ZPD.

error: Invalid argument Type. This module will throw an exception if the user sets the parameter apodType to a value other than "prePhaseCorr" or "postPhaseCorr".

error: Invalid argument name. This module will throw an exception if the user sets the parameter apodName to a value that refers to an apodizing function which is not available. A list of valid apodizing functions is printed to the console.

6.15. Producing Spectra from the Interferograms (Fourier Transform)

6.15.1. Module Description

The purpose of the *Fourier Transform* task is to transform the set of interferograms from a SPIRE spectrometer observational building block into a set of spectra. This processing task can transform both double-sided and single-sided interferograms. **Double-sided Transform.** For the double-sided transform, each interferogram in the SDI is examined and only the double-sided portion of the interferogram, where data are available between $-OPD_{max}$ and $+OPD_{max}$, is used to compute the resultant spectrum. The resultant spectra will contain both real and imaginary components. **Single-sided Transform.** For the single-sided transform, only those interferogram samples to one side of the position of zero path difference (ZPD) are considered. The spectra that result from the single-sided transform contain only real components as perfect even symmetry with respect to ZPD is assumed.

6.15.2. Input Data Products

SDI **Spectrometer Detector Interferogram** This product contains interferograms for each spectrometer detector for each scan of the observational building block.

6.15.3. Output Data Products

SDS **Spectrometer Detector Spectrum** The output SDS data product contains the calculated spectrum for each of the interferograms in the input SDI.

6.15.4. Input Calibration Products

No calibration data are needed.

6.15.5. How to use the *Fourier Transform* Module

6.15.6. Parameter Options

Users can set the following parameters of the Fourier Transform module:

ftType This parameter defines the type of Fourier Transform that is to be performed - either double-sided or single-sided. The distinction is determined by whether the task is run before the phase correction step, or after it. Therefore, the possible values of "ftType" are "prePhaseCorr" and "postPhaseCorr". If apodization is applied directly before the Fourier Transform, the value of ftType must equal the value of apodType in the apodizeIfgm task.

In the "prePhaseCorr" mode, the portion of the input interferograms that is symmetric about the position of zero path difference (OPD = {-N/2 ..., 0, ..., N/2}) will be used to compute the output spectra.

In the "postPhaseCorr" mode, only the portion of the input interferograms that is greater than or equal to the position of zero path difference (OPD = {0, ..., N/2}) will be used to compute the output spectra.

The data type of the flux columns in the output product depends on the value chosen for this parameter. The flux column will be of type

	`Complex1d` if "ftType" is set to "prePhaseCorr". If "ftType" is set to "postPhaseCorr", then the flux column will be of type `Double1d`.
zeroPad (Optional)	This parameter defines how the interferograms are zero-padded before transformation. The possible values are "None", "standard" and "2**n". If the parameter is set to "None", no zeros are appended to the interferogram. If the parameter is set to "standard", the standard zero-padding is applied, to give four samples per resolution element in the final spectra. This will ensure that the interval of the wavenumber grid is 0.01/0.05/0.25 cm^{-1} for data collected in high/medium/low resolution mode. If the parameter is set to "2**n", then zeros are appended to the interferograms to provide exactly 2^n data points to the Fast Fourier Transform routine to minimize execution time.

6.15.7. Examples

The following are examples of the SPIRE spectrometer Fourier transform task.

```
from herschel.spire.ia.pipeline.spec.ft import FourierTransformTask
dsds=fourierTransform(sdi, ftType="prePhaseCorr", zeroPad="None")
ssds=fourierTransform(sdi, ftType="postPhaseCorr", zeroPad="standard")
```

6.15.8. Error messages

error: Parameter ftType must be set. The parameter ftType is mandatory and must be set by the user.

error: Delta not a constant. The module throws an exception if the OPD grid is not equidistant. The result from this module would be incorrect if the calculation went ahead regardless.

error: Zero not found in array. The module throws an exception if there is no zero in the calculated OPD array.

error: Array does not have positve and negative values. The module throws an exception if there is no zero in the calculated OPD array.

error: Array lengths are not equal.

error: Array values differ at index "xx".

error: ftType not set to an accepted value.

error: ftType value does not match the apodType set in input SDI.

6.16. Phase Correction

6.16.1. Module Description

The *Phase Correction* module corrects for any asymmetries in the interferogram about the position of Zero Path Difference (ZPD). The symmetry of the optical layout of a Fourier transform spectrometer (FTS) theoretically implies that the recorded interferograms also exhibit even symmetry. The Fourier

transform of an evenly symmetric interferogram contains only real components. The presence of dispersive elements and the possibility that the position of ZPD is not being sampled can result in an interferogram with signal samples that are not symmetric about ZPD. The resulting spectrum will contain both real and imaginary components and therefore a non-zero phase. Phase correction proceeds in two steps: the first characterizes the phase of the measured interferogram; the second removes the phase.

6.16.2. Input Data Products

SDI — **Spectrometer Detector Interferogram** The SDI product contains interferograms for each spectrometer detector for each scan of the observational building block.

SDS — **Spectrometer Detector Spectrum** The SDS product contains the double-sided (low resolution) spectra for each spectrometer detector for each scan of the observation.

6.16.3. Output Data Products

SDI — **Spectrometer Detector Interferogram** The output SDI product contains phase-corrected interferograms for each spectrometer detector for each scan of the observational building block.

SDS — **Spectrometer Detector Spectrum** The set of double-sided spectra SDS that was provided as an input to this processing module will be modified such that its double-sided spectra will be phase-corrected.

6.16.4. Input Calibration Products

nlp (Optional) — **Non-Linear Phase** This calibration product describes that portion of the phase which is stable over time because it is based on instrumental characteristics. The non-linear phase will be characterized from in-flight data for each detector in SLW and SSW and made available as calibration data which can then be taken out of each interferogram measured.

phaseCorrLim (Optional) — **Phase Correction Limits** This optional calibration product indicates where the pipeline expects the spectral SNR to be at its maximum in order to be suitable for a fit to the phase. The spectral ranges are given in units of cm^{-1} and correlate roughly to the measured spectral bands for each spectrometer detector.

6.16.5. How to use the *Phase Correction* Module

6.16.6. Control parameters

Users can set the following parameters of the Phase Correction module:

polyDegree (Optional) — The optional parameter "polyDegree" sets the degree of the polynomial function that is used to fit the measured in-band phase. The parameter can be set to any positive integer. If "polyDegree" is set to unity for a linear fit, then phase correction can only shift ZPD, the point of symmetry of the interferogram, and, in the process, change where the interferogram is sampled. However, phase correction with a linear phase fit cannot change the shape of the interferogram. If "polyDegree" is set to values of two or higher then the underlying shape of the interferogram can change as well. The parameter has to be selected with care in the presence of significant noise. In this case,

polynomials with a order greater than 4 can easily fit to random noise rather than a stable and repeatable instrumental phase. The default value is 2.

convolApodName (Optional) — In the case of phase-correcting a high resolution interferogram, the derived Phase Correction Function (PCF) is convolved with the data. In order to avoid artifacts from the convolution with an array that has non-zero values at its edges, the PCF can be apodized. A range of apodizing functions is available. The optional parameter "convolApodName" can be set to one of the values specified below. Care should be taken to select an apodizing function with values close to zero at its edges. It is a non-trivial decision to select the best apodizing function for phase correction. At the same time, the impact of the apodization function on the quality of the phase correction is limited. By default, no apodizing function is applied.

The following apodizing functions are available:

- "aGauss_19", Gaussian
- "aHM_15", Hamming
- "aHN_17", Hanning
- "aNB_11" - "aNB_20", ten adjusted Norton Beer functions

pcfSize (Optional) — This optional parameter specifies the number of elements in the phase correction function (PCF) which is used to correct single-sided interferograms, e.g. for high resolution data. A benefit of a larger PCF is that it may be able to correct for stronger asymmetries (further away in terms of OPD). However, a larger PCF will also lead to a stronger reduction in spectral resolution because the convolution operation invalidates points at the interferogram edges. The convolution will invalidate a number of points on either side of the interferogram which is equal to half the value specified by "pcfSize". The default value is 127.

6.16.7. Examples

Assuming a spectrometer detector interferogram product `sdi` and an observation context `obs`:

```
from herschel.spire.ia.pipeline.spec.phase import PhaseCorrectionTask
dsds = fourierTransform(sdi, ftType="prePhaseCorr", zeroPad="None")
sdi = phaseCorrection(sdi, \
       sds=dsds, \
       polyDegree=2, \
       convolApodName="aNB_20", \
       pcfSize=127, \
       phaseCorrLim=obs.calibration.spec.phaseCorrLim, \
       nlp=obs.calibration.spec.nlp)
```

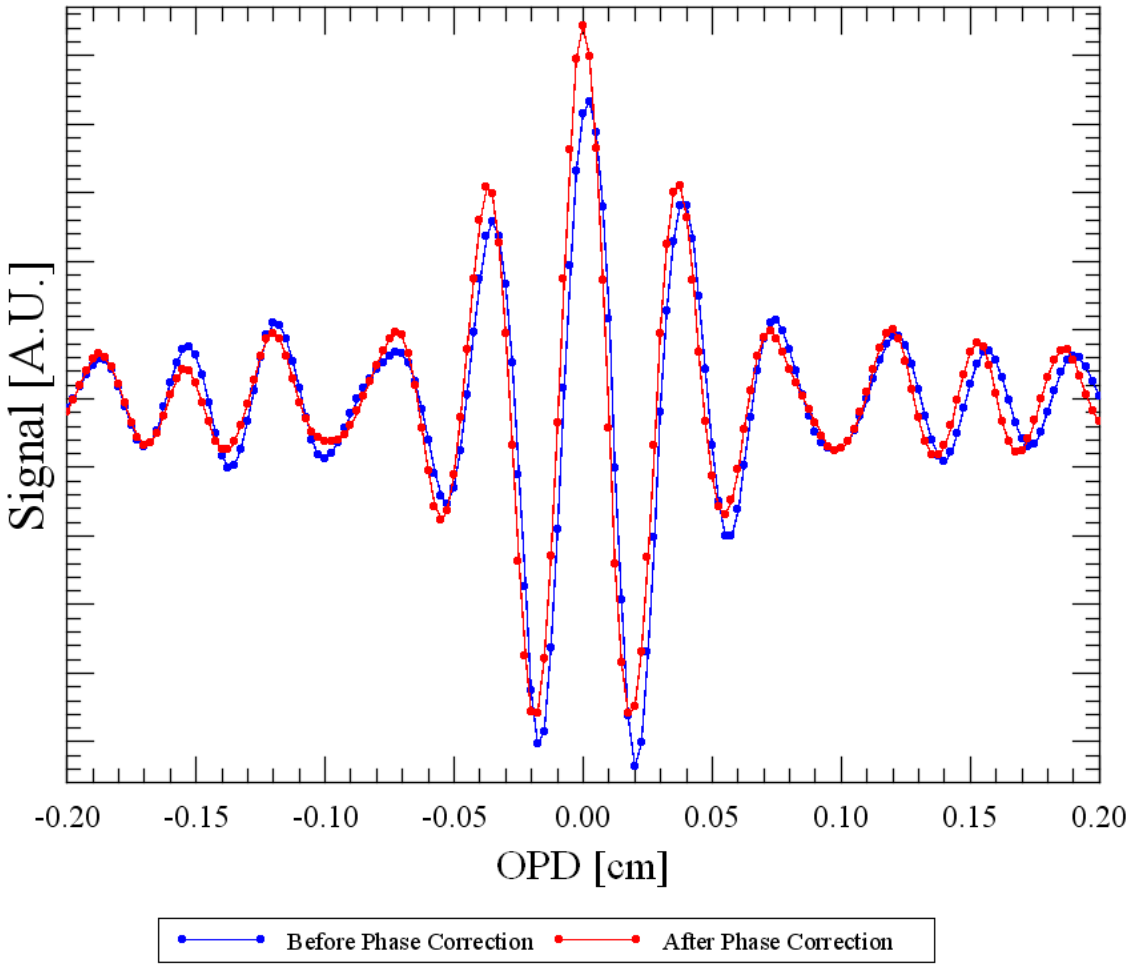

Figure 6.14. An example for phase-correction: The original interferogram (blue) shows a slight displacement from ZPD and asymmetries with respect to the center burst. Note how the first sidelobes at plus and minus 0.02 cm have different amplitudes. The phase-corrected interferogram (red) is well centered about ZPD and its first sidelobes have very similar amplitudes.

6.16.8. Error messages

error: No match for "xx" found in SDS. Cannot continue. This module will throw an exception if the low resolution SDS product does not contain a scan contained in the input SDI product.

error: No match for "xx" found in SDS. Cannot continue. This module will throw an exception if the low resolution SDS product does not contain a detector contained in the input SDI product.

error: Sdi has no "type" parameter in MetaData. This module will throw an exception if the SDI product does not contain a valid "type" entry in its metadata.

error: Sdi has no "numScans" parameter in MetaData. This module will throw an exception if the SDI product does not contain a valid "numScans" entry in its metadata.

error: Sdi has no "aot" parameter in MetaData. This module will throw an exception if the SDI product does not contain a valid "aot" entry in its metadata.

error: Sdi has no "commandedResolution" parameter in MetaData. This module will throw an exception if the SDI product does not contain a valid "commandedResolution" entry in its metadata.

error: Unknown Apodization Function. This module will throw an exception if the user-specified apodizing function is not available.

error: Output SDI is empty.

error: Cannot get double sided portion: Range of OPD grid does not contain 0.

error: Cannot get double sided portion: ZPD cannot be determined.

6.17. Averaging Spectra to Produce One Final Spectral Product

6.17.1. Module Description

The *Average Spectra* module computes, on a wavenumber-by-wavenumber basis for each spectrometer detector, the average and the uncertainty of the spectral intensities across all scans. The average is calculated as the arithmetic mean of the spectral components. The uncertainty is calculated as the standard deviation of the spectral components. The module allows to average all scans or to keep scans from different directions separate. The module rejects outliers by default. The module removes data outside of the optical passband by default in order to reduce data volume.

6.17.2. Input Data Products

SDS — **Spectrometer Detector Spectrum** The input SDS product contains spectra for each spectrometer detector for each scan of the observational building block.

6.17.3. Output Data Products

SDS — **Spectrometer Detector Spectrum** The output SDS data product contains the average flux density and uncertainty for each spectrometer detector. There will be one scan in the output SDS.

6.17.4. Input Calibration Products

bandEdge — The **Spectrometer Band Edge** calibration product contains the optical passband for each detector.

6.17.5. How to use the *Spectral Averaging* Module

6.17.6. Parameter Options

Users can set the following parameters of the Spectral Averaging module:

separateScanDirections (Optional) — The optional "separateScanDirections" parameter allows users to average spectra which were recorded while the mirror stage mechanism SMEC was traveling in a specific direction. If "separateScanDirections" is set to its default value "False", then the output product will contain only one composite dataset with data derived from scans in both directions. If "separateScanDirections" is set to "True", then the output product will contain two composite datasets with data averaged separately for forward and reverse scans.

outlierType (Optional)	The averaging task will ignore outliers while computing the spectral average and standard deviation if this optional parameter is set to one of its allowed values "3Sigma", "STD", or "MAD". Statistical outliers are defined as values outside of an acceptable range. Acceptable ranges are defined as follows: the arithmetic mean plus or minus three times the standard deviation for "3Sigma"; the arithmetic mean plus or minus a threshold factor (see next keyword) times the standard deviation for "STD"; the median plus or minus a threshold factor (see next keyword) times the median absolute deviation for "MAD".
thresholdFactor (Optional)	The user can define the threshold factor if the outlierType is "STD" or "MAD". The real value of the optional parameter "thresholdFactor" must be positive. The default value is 3.2905 if the outlierType is "STD" and 4.8785 if the outlierType is "MAD", ensuring that about 0.1% of all samples are rejected in randomly distributed data.
INCLUDE_OOB (Optional)	The user can set this boolean keyword to True to retain the spectral data outside of the optical passband of each detector. By default, the task will delete the out-of-band data in order to reduce data volume.

6.17.7. Examples

Assuming `ssds` is an SDS product:

```
from herschel.spire.ia.pipeline.spec.average import AverageSpectraTask
asds = averageSpectra(sds=ssds, bandEdge=bandEdge, \
                      separateScanDirections=False, \
                      outlierType="STD", thresholdFactor=3.2905, \
                      INCLUDE_OOB=Boolean.True)
```

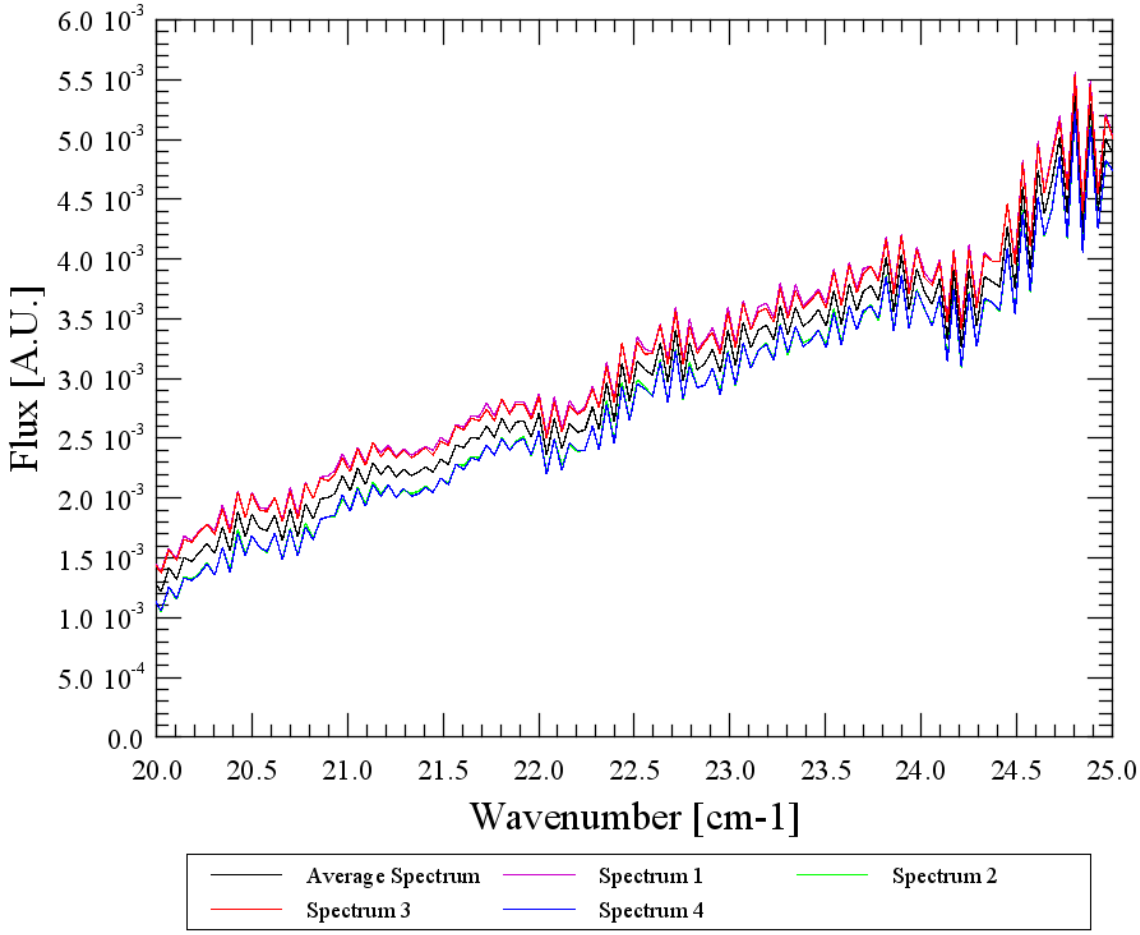

Figure 6.15. The spectra from four scans (magenta, green, red, blue) are averaged to yield one final spectrum (black).

6.17.8. Error messages

error: "xx" spectral length mismatch. Scan "xx": length = x. Scan "yy": length = y. For each detector, the task verifies that the wavenumber grids are of identical length for all scans. An exception will be thrown if they differ in length.

error: "xx" Wavenumber column mismatch at index "i". Scan "xx": wn = x. Scan "yy": wn = y. For each detector, the task verifies that the wavenumber grids are identical for all scans. An exception will be thrown if they differ by more than a small number which is below the precision of a variable of type `Float`.

error: Unsupported Flux type. The module throws an exception if the data type of the flux column is neither Float nor Double.

error: Input SDS contains no scans.

error: Channel "xx" has no "yy" direction scans. In order to separate scan directions, at least one scan is required in either direction.

error: Ra units among SpireSpectrum1d don't match.

error: Dec units among SpireSpectrum1d don't match.

error: Threshold is negative.

6.18. Spectrum Flux Calibration

6.18.1. Module Description

The *Flux Conversion* module performs the absolute flux calibration for the SPIRE spectrometer pipeline. Based on measurements of the telescope, the module applies an extended source flux calibration. It can also be used to convert the flux calibration for extended sources to the flux calibration for point sources. The latter is based on a characterization of the beam shape, and a deep measurement and detailed model of Uranus. The values in the Spectrometer Detector Spectrum (SDS) product are converted from Volts per cm^{-1} to units relating to the flux density of the source.

6.18.2. Input Data Products

SDS — **Spectrometer Detector Spectrum** The input SDS product contains a spectrum for each detector for each scan of the observational building block.

6.18.3. Output Data Products

SDS — **Spectrometer Detector Spectrum** The output SDS product contains a flux-calibrated spectrum for each spectrum contained in the input SDS. The signal values are now in units that relate directly to the observed astronomical source.

If the user chooses to apply a point source flux calibration the output product is a SpectrometerPointSourceSpectrum, a slightly different type of SDS product.

6.18.4. Input Calibration Products

fluxConv — The **Spectrometer Flux Conversion** calibration product contains for each detector a wavenumber-dependent calibration table to relate V per wavenumber to units of flux density per wavenumber. Flux calibration depends on whether the nominal and bright detector settings were used. The calibration may change over time. It is therefore necessary to retrieve the right calibration product from the calibration context by indicating when the observation was made and whether nominal or bright detector settings were used.

beamParam — The **Spectrometer Beam Parameters** calibration product contains the wavenumber-dependent point-source flux conversion factors for the central detectors of each spectrometer array.

6.18.5. How to use the *Flux Conversion* Module

This module performs two different flux conversions: By default, the flux calibration for extended sources is applied. In addition, the module can also be used to convert from extended to point source flux calibration.

6.18.6. Parameter Options

Users can set the following parameter of the Spectral Flux Conversion module:

APPLY_POINT_SOURCE (Optional) — The boolean parameter "APPLY_POINT_SOURCE" allows users to convert from flux calibration for extended sources to flux calibration for point sources. By default, this parameter is set to False.

6.18.7. Examples

Assuming `sds` is a spectrometer detector spectrum product with the signal in units of volts per cm^{-1} and `obs` an observation context:

```
  from herschel.spire.ia.pipeline.spec.fluxconv import SpecFluxConversionTask
  # get the bias mode (biasMode) from the SDS
  biasMode = sds.meta["biasMode"].value
  # get the observation start date from the SDS
  startDate = sds.startDate
  # get the correct edition of the flux conversion calibration product
  fluxConv = obs.calibration.spec.fluxConvList.getProduct("HR", -"unapod",
biasMode, startDate)
  # apply the extended flux conversion:
  sds = specFluxConversion(sds, fluxConv=fluxConv)
  # get the correct edition of the beam profile calibration product
  beamParam = obs.calibration.spec.beamParamList.getProduct(Boolean.FALSE,
biasMode, startDate)
  # apply the point source flux conversion:
  pointSourceSds = specFluxConversion(sds=sds, fluxConv=fluxConv, \
             beamParam=beamParam, APPLY_POINT_SOURCE=Boolean.TRUE)
```

The SOF1 pipeline script gives further indication of how to retrieve the correct calibration product for a specific SDS.

6.18.8. Error messages

There are no specific error messages defined for this task.

6.19. Removing any Optical Crosstalk from the Spectra

6.19.1. Module Description

The *Remove Optical Crosstalk* module accounts for effects of the optical chain of the SPIRE instrument which lead to a detector receiving radiation that should have been received by another detector.

6.19.2. Input Data Products

SDS — **Spectrometer Detector Spectrum** The input SDS product contains spectra for each detector.

6.19.3. Output Data Products

SDS — **Spectrometer Detector Spectrum** The output SDS product contains spectra for each detector.

6.19.4. Input Calibration Products

optCross — The **Spectrometer Optical Crosstalk** calibration product contains two matrices which describe the optical crosstalk for the detectors of

the SLW and SSW detector array respectively. Currently, no systematic, linear cross-talk has been identified and this processing task is used as a placeholder.

6.19.5. How to use the *Remove Optical Crosstalk* Module

6.19.6. Parameter Options

The optical crosstalk correction module does not use any parameters that can be controlled by the user at run-time.

6.19.7. Examples

Assuming `ssds` is an SDS product and `obs` an observation context:

```
from herschel.spire.ia.pipeline.spec.optcross import SpecOptCrossCorrectionTask
ssds = specOptCrossCorrection(ssds, optCross=obs.calibration.spec.optCross)
```

6.19.8. Error messages

There are no specific error messages defined for this task.

6.20. Preprocessing for Spatial Regridding

6.20.1. Module Description

The purpose of *preprocessing for spatial regridding* is to create data that is in a format which may be used by `NearestNeighbourProjectionTask` to create a spectral cube. Preprocessing for spatial regridding performs three functions:

1. It creates instrument-independent representations of spectral data. Data within Spectrometer Detector Spectrum (SDS) objects are copied and formatted into a set of equivalent 3-dimensional arrays.

2. It verifies that spectra are compatible, that they may be used to create a spectral cube.

3. It separates data from SLW and SSW detector arrays into two distinct objects.

`SpirePreprocessCubeTask` preprocesses data for spatial regridding. It accepts an array of SDS objects as input and returns a `SpirePreprocessedCube` from which one may extract 3-dimensional arrays that are equivalent to the SDS input. The 3-dimensional arrays may then be used as input for the spatial regridding module to create spectral cubes.

6.20.2. Input Data Products

`SpirePreprocessCubeTask` requires either an SDS object or a python array of SDS objects as input:

SDS — **Spectrometer Detector Spectrum.** The `SpirePreprocessCubeTask` can take in a single SDS that contains spectra from the SLW and SSW detector arrays. The SDS may have more than one scan.

[SDS] — **A python array of SDS objects.** For successful processing, each spectrum in the array must be the same size and have the same wavenumber grid. In addition, the units of all spectra must be the same.

6.20.3. Output Data Products

SPPC — **Spire Preprocessed Cube.** The output SPPC is a container for flux, error, flags, RA and Dec data. The data of the output SPPC are represented as 3-dimensional arrays and are equivalant to the data of the input SDS. Depending on the `arrayType` input parameter, the output SPPC may contain spectra from the SLW array, SSW array or both arrays.

6.20.4. Input Calibration Products

No calibration data are needed.

6.20.5. How to use the *Preprocessing for Spatial Re-gridding* Module

`SpirePreprocessCubeTask` will only create `SpirePreprocessedCube` objects from data with identical wavenumber grids - if two spectra have different wavenumber grids, they cannot be contained by the same preprocessed cube. The current data processing pipelines produce SLW and SSW spectra with different wavenumber grids. This means that for any given SDS (or array of SDSs), `SpirePreprocessCubeTask` needs to be run twice: once to create a `SpirePreprocessed-Cube` containing SLW spectra and once to create a `SpirePreprocessedCube` containing SSW spectra. You may create a `SpirePreprocessedCube` of either SLW or SSW spectra by specifying `arrayType="SLW"` or `arrayType="SSW"` as a task parameter in `SpirePreprocess-CubeTask`.

If, during interactive analysis, you want to create a `SpirePreprocessedCube` with both SLW and SSW spectra, you will need to ensure that all spectra are of identical size and have the same wavenumber grid.

6.20.6. Parameter Options

The following parameters function as data filters. They specify properties (array type, detector name, unvignetted/vignetted) which determine which spectra may be passed from input to output. For example, filters may be combined to produce output that consists solely of data from unvignetted SLW detectors `(UNVIGNETTED=Boolean.TRUE,arrayType="SLW")` or to output specific detectors `(detectors =[SLWA1,SLWC3,SLWB4])`.

arrayType (Optional) — This option can have one of two values: "SLW" or "SSW". This parameter functions as a filter by specifying the detector array of the detectors that *may* be passed to output. The detectors in the specified detector array pass through this filter - but they may be filtered out by other filters. If `arrayType="SLW"`, the SLW spectra pass through this filter. If `arrayType="SSW"`, the SSW spectra pass through this filter. If `arrayType` is not given as a task parameter, no filtering by array type occurs.

detectors (Optional) — A python array of detector names. Only detectors named in the `detectors` list *may* be passed to output. The `detectors` parameter functions as a filter, removing all detectors not named on its list. The

	detectors specified by `detectors` pass through the `detectors` filter - but they may be filtered out by other filters. If `detectors` is not given as a task parameter, no filtering by detector name occurs.
UNVIGNETTED (Optional)	A boolean value. If `UNIVIGNETTED` is true, only unvignetted detectors *may* be passed to output. That is, if `UNIVIGNETTED` is true, only unvignetted detectors "pass through" the `UNVIGNETTED` filter - but they may be filtered out by other filters. If `UNIVIGNETTED` is false or not specified, there is no filtering that is based upon vignetting. A detector is defined to be vignetted if it is located on the outer ring of a detector array. Unvignetted detectors are not located on the outer ring of a detector array. For example, SLWC2, SLWC3, SLWC4 are unvignetted detectors whereas SLWC1 and SLWC5 are vignetted detectors that would be filtered out if `UNIVIGNETTED=Boolean.TRUE`.

6.20.7. Examples

Assuming `sdsArray` is an array of SDS objects, the following code creates a `SpirePreprocessedCube` for SLW detectors and a `SpirePreprocessedCube` for SSW detectors:

```
    # Instantiate task
    sppct = SpirePreprocessCubeTask()

    # Create SLW preprocessed cube
    spcSlw =sppct(sdsList=sdsArray,arrayType="SLW")
    # Create SSW preprocessed cube
    spcSsw =sppct(sdsList=sdsArray,arrayType="SSW")
```

To create a `SpirePreprocessedCube` with unvignetted SLW detectors:

```
     # Create preprocessed cube from unvignetted SLW detectors
     spcSlwUnvignetted
=sppct(sdsList=sdsArray,arrayType="SLW",UNVIGNETTED=Boolean.TRUE)
```

To create a `SpirePreprocessedCube` from SSWA1, SSWA2, SSWA3 and SSWA4:

```
     # Create preprocessed cube with SSWA1, SSWA2, SSWA3 and SSWA4
     spcSswA =sppct(sdsList=sdsArray,detectors=["SSWA1","SSWA2","SSWA3","SSWA4"])
```

6.20.8. Error messages

error: Cannot make a SpirePreprocessedCube without eligible channels. An exception is thrown if the input SDS is empty or is filtered so that there are no channels to process.

error: The `unitType` unit of `spectrum1` and `spectrum2` do not match. An exception is thrown if any units of the input spectra do not match.

error: Wave column length between `channelName1` and `channelName2` is inconsistent. An exception is thrown if the lengths of the wavenumber grids of any two spectra are different.

error: Wave column value of `channelName1` and `channelName2` at index `i` is inconsistent. An exception is thrown if any two spectra have different wavenumber grids.

6.21. Producing Spectral Cubes (Spatial Re-gridding)

6.21.1. Module Description

The purpose of the *spatial regridding module* is to create cubes from spectra. Given a set of spectra as input, the spatial regridding module returns an equidistantly gridded spectral cube. The spectral cube consists of a stack of images where each image represents the spatial distribution of flux within a particular wavebin.

Spectral cubes can be created from three different sets of input:

1. A set of 3-dimensional arrays that specify flux, RA, Dec, flags, and error. Flux, RA, and Dec are required. Flags and error are optional.

   For each 3-dimensional array:

   - The first dimension is the spectral dimension.
   - The second dimension is degenerate for the SPIRE spectrometer and has a range of 1.
   - The third dimension specifies the detector.

   All 3-dimensional arrays must have the same dimensions. Elements from different arrays having the same indices correspond to each other. For example, the flux at `flux[2,0,2]` corresponds to the error at `error[2,0,2]`.

2. A spectral cube. The module allows you to regrid a spectral cube --to create a new cube having different dimensions and a different coordinate system.

3. A python array of spectral cubes. The input spectral cubes are combined into a single spectral cube.

In addition, for all cases, the dimensions of the output cube and the world coordinate system must be specified by a target grid parameter.

`NearestNeighbourProjectionTask` regrids by copying data from the closest spectrum of each pixel to the output `SpectralSimpleCube`. The "nearest neighbour" of the pixel, as determined by examining the angular distance between the pixel and the RA and Dec values of each spectrum, *is* the output at that pixel.

6.21.2. Input Data Products

Spectral cubes can be created from three different sets of input:

1. flux (Mandatory)	**Flux** This is a 3-dimensional array (`Double3d`) of flux from either SLW or SSW detectors.
ra (Mandatory)	**Right Ascension** This is a 3-dimensional array (`Double3d`) of right ascension values associated with an array of detectors (SLW or SSW). As SPIRE spectrometer pointings do not vary with wavenumber, elements of the RA parameter only vary in the detector dimension.
dec (Mandatory)	**Declination** This is a 3-dimensional array (`Double3d`) of declination values associated with an array of detectors (SLW or SSW). As SPIRE spectrometer pointings do not vary with wavenumber, elements of the dec parameter only vary in the detector dimension.

2. SSC (Mandatory) — **Spectral Simple Cube** This product is a stack of images at different frequencies. Each image has the same height and width. Embedded within the cube is a world coordinate system which specifies the relationship between pixel coordinates and world coordinates.

3. [SSC] (Mandatory) — **Spectral Simple Cube List** This is a python array of spectral simple cubes. When supplied with a list of spectral simple cubes as input, the spatial regridding module uses the flux values of the cubes to create an output cube containing new images and a new world coordinate system. Each spectral cube in the array must have the same units.

6.21.3. Output Data Products

SSC — **Spectral Simple Cube** The output SSC contains spectral data on an equidistant RA/Dec grid.

6.21.4. Input Calibration Products

No calibration data are needed.

6.21.5. How to use the *Spatial Regridding* Module

To create a spectral cube from SDS objects, you will first need to create `SpirePreprocessedCube` objects using `SpirePreprocessedCubeTask` (see Preprocessing for Spatial Regridding). The next step is to extract 3-dimensional arrays of flux, error, RA and Dec from a `SpirePreprocessedCube` and use them as input parameters for `NearestNeighbourProjectionTask`.

To process the 3-dimensional arrays of data, pass the flux, RA, Dec, error and flag arrays as parameters into `NearestNeighbourProjectionTask`, which performs the regridding. A `TargetGrid` parameter also needs to be passed to the task. If you do not have a suitable `TargetGrid` for your data, create one using the `targetGrid` method supplied by `NearestNeighbourProjectionTask`. The units of the `TargetGrid` parameter will determine the spectral grid units of the output SSC. If you wish to specify a flux unit for the output cube, use the `fluxUnit` parameter.

`NearestNeighbourProjectionTask` requires that all RA and Dec input be measured in degrees. Units for quantities other than RA and Dec make no difference to calculations -they are only passed to the output SSC

6.21.6. Parameter Options

You can set the following parameters of the Spatial Regridding module:

error (Optional) — **Error on flux** This is a 3-dimensional array (`Double3d`) of error values on the flux from either SLW or SSW detectors.

flag (Optional) — **Flag** This is a 3-dimensional array (`Double3d`) of flags associated with either SLW or SSW detectors. Flags may specify additional information about the processing history and status of flux values.

fluxUnit(Optional) — **Flux unit** This parameter describes the flux unit of the input flux. The spectral cube output will have the same flux unit.

meta (Optional) — **Metadata** Metadata contains additional information that is passed to the spectral cube output.

target (Mandatory) — **Target Grid** The target grid defines the shape of the output spectral cube. It defines:

- The number rows and columns in each image of the output spectral cube.
- The number of images and the wavenumbers associated with each image of the output spectral cube.

It also defines the projection used to convert between world coordinates and pixel coordinates. The default `TargetGrid` created by `NearestNeighbourProjectionTask` specifies gnonomic projection.

6.21.7. Examples

Assuming `sdsArray` is an array of SDS objects, the following code creates a `SpectralSimpleCube` from the unvignetted SSW detectors in the array:

```
      from herschel.ia.toolbox.spectrum.projection import
NearestNeighbourProjectionTask

      # Create preprocessed cube
      spirePreprocessCube = SpirePreprocessCubeTask()
      spc = spirePreprocessCube(sdsList=sdsArray, arrayType = -"SSW", UNVIGNETTED
= Boolean.TRUE)
      # Extract data for regridding
      flux = spc.getFlux()
      error = spc.getError()
      ra = spc.getRa()
      dec = spc.getDec()
      flag = spc.getFlag()
      wave = spc.getWave()
      fluxUnit = spc.getFluxUnit()
      meta = spc.getMeta()

      # Create and initialize task
      nnpt = NearestNeighbourProjectionTask()
      nnpt.setWaveUnit(spc.getWaveUnit()) # specifies wavenumber grid unit for
TargetGrid creation
      # Get target grid for data.
      grid = nnpt.targetGrid(ra,dec,wave)
      # Create SSW spectral cube

sc=nnpt(flux=flux,error=error,flag=flag,ra=ra,dec=dec,target=grid,fluxUnit=fluxUnit,meta=meta)
```

Assuming `cube` is a `SpectralSimpleCube` object and `targetGrid` is a `TargetGrid`, the following code creates a `SpectralSimpleCube` by regridding the input cube:

```
      from herschel.ia.toolbox.spectrum.projection import
NearestNeighbourProjectionTask

      # Create task
      nnpt = NearestNeighbourProjectionTask()
      # Regrid a spectral cube
      sc = nnpt(cube=cube,target=targetGrid)
```

Assuming `cubeList` is an array of `SpectralSimpleCube` objects and `targetGrid` is a `TargetGrid`, the following code creates a `SpectralSimpleCube` by combining the input cubes:

```
      from herschel.ia.toolbox.spectrum.projection import
NearestNeighbourProjectionTask

      # Create task
      nnpt = NearestNeighbourProjectionTask()
```

```
# Create spectral cube from cubes in list
# All cubes in the list must have identical units!
sc = nnpt(cubeList=cubeList,target=targetGrid)
```

6.21.8. Error messages

error: Invalid arguments This module will throw an exception unless:

- The input parameters of `flux`, `ra`, `dec` and `target` are supplied (and `cube` and `cubeList` are not).

or

- The `cube` or `cubeList` parameter is supplied, and the `target` parameter is supplied (and `flux`, `ra`, and `dec` are not).

error: The SpectralSimpleCube list must contain at least one cube. An exception is thrown if the list defined by the `cubeList` parameter is empty.

error: All cubes in the list must have equal spectral depth. An exception is thrown if the cubes in the `cubeList` parameter do not all have the same wavenumber grid.

error: If one cube has error, then so must all of them. An exception is thrown if a cube in the `cubeList` parameter has error values but not all cubes in the list have error values.

error: If one cube has flag, then so must all of them. An exception is thrown if a cube in the `cubeList` parameter has flag values but not all cubes in the list have flag values.

error: The given world coordinate system returned NaN for pixel `row`, `column`. An exception is thrown if an invalid world coordinate system was embedded in one of the `SpectralSimpleCube`s in the `cubeList` parameter.
