SUBJECT: **SPIRE FTS Line Source Analysis for Cross-Calibration**

PREPARED BY: **Peter Imhof**

DOCUMENT No: **SPIRE-BSS-REP-003269**

ISSUE: **Issue 1.2** **Date:** **22 May 2012**

APPROVED BY: **Date:**

Distribution

Name

Bruce Swinyard
David Naylor
Edward Polehampton
Elena Puga
Ivan Valtchanov
Matt Griffin
Matthijs van der Wiel
Nanyao Lu
Ros Hopwood
Tanya Lim
Trevor Fulton

Change Record

ISSUE	DATE	Changes
Draft 0.1	19 December 2011	First draft
Draft 0.2	20 December 2011	Corrections after input from MvdW
Draft 0.3	12 January 2012	Add data on AFGL4106, source velocities Corrections after input from RH Updated scripts
Issue 1.0	09 February 2012	Fix entry in Table 3 – no other change.
Issue 1.1	12 February 2012	Removed 0x5000C542 from Table 5.
Issue 1.2	22 May 2012	Update to include observations up to OD1032

TABLE OF CONTENTS

Glossary

Acronym	Description
CR	Calibration Resolution
CVS	Concurrent Versions System
FTS	Fourier Transform Spectrometer
HIPE	Herschel Interactive Processing Environment
HR	High Resolution
HSA	Herschel Science Archive
LSR	Local Standard of Rest
OD	Operational Day
SPG	Standard Product Generation
SPIRE	Spectral and Photometric Imaging REceiver

1. INTRODUCTION

The purpose of this document is to describe how the SPIRE spectrometer data of routine calibration line sources were processed and analyzed.

1.1 Documents

1.1.1 Applicable Documents

Number	Document Title	Document Number	Issue
AD01	SPIRE Spectrometer Pipeline Description	SPIRE-BSS-DOC-002966	3.1
AD02	SPIRE Routine Phase Calibration Plan	SPIRE-RAL-DOC-003131	1.0
AD03			
AD04			

1.1.2 Reference Documents

Number	Document Title	Author(s)	Issue
RD01			
RD02			

2. DATA

The SPIRE Routine Phase Calibration Plan (AD02) defines three point-like sources outside of the Solar System with prominent line features to be observed on a regular basis as routine calibration sources for point source observations. These sources are as per section 4.2.4.5 of AD02:

1. **NGC 7027**
   RA = 316.756625 deg, Dec = 42.2361666667 deg
   RA = 21:07:01.590, Dec = 42:14:10.200
2. **CRL 618**
   RA = 70.7235 deg, Dec = 36.1148333333 deg
   RA = 04:42:53.640, Dec = 36:06:53.400
3. **AFGL 2688**
   RA = 315.57825 deg, Dec = 36.6947777778 deg
   RA = 21:02:18.780, Dec = 36:41:41.200

The SPIRE ICC has also added the following point-like sources to its routine calibration program:

4. **NGC 6302**
   RA = 258.434208333 deg, Dec = -37.1044166667 deg
   RA = 17:13:44.210, Dec = -37:06:15.900
5. **AFGL 4106**
   RA = 155.831125 deg, Dec = -59.53469444444444 deg
   RA = 10:23:19.470, Dec = -59:32:04.900
6. VY CMa
7. CW Leo (also known as IRC+10216)

Dark Sky and Orion Bar have been selected as the main sources for the routine calibration program of jiggle observations.

To date, the following routine observations were made with nominal detector settings of the sources mentioned above under points 1 to 5:

#	*OD*	*Obsid*	*Number of repetitions*	*Spectral resolution*
1	217	0x50002aa5	10	CR
2	227	0x50002c7e	10	CR
3	240	0x50002e41	17	HR
4	240	0x50002e44	17	HR
5	240	0x50002e45	10	CR
6	250	0x50002fe7	10	CR
7	326	0x50004094	5	CR
8	342	0x50004693	5	CR
9	368	0x50004b86	5	CR
10	383	0x50004eee	5	CR
11	404	0x50005489	5	CR
12	557	0x50007f40	4	CR
13	572	0x5000832a	4	CR
14	601	0x500088e4	4	CR
15	711	0x5000a533	4	CR
16	742	0x5000ad81	4	CR
17	908	0x5000d5b9	4	CR
18	971	0x5000e94f	4	CR
19	988	0x5000ee25	4	CR

Table 1: Observations of NGC7027 after OD189; observations marked in yellow were not made at the nominal sky position.

#	*OD*	*Obsid*	*Number of repetitions*	*Spectral resolution*
1	275	0x50003459	5	CR
2	288	0x5000366e	17	HR
3	288	0x5000366f	5	HR
4	302	0x50003a32	2	CR
5	317	0x50003cc5	2	CR
6	451	0x50006193	4	CR
7	466	0x50006877	4	CR
8	495	0x50006bff	4	CR
9	655	0x500092ca	4	CR
10	682	0x50009ab8	4	CR
11	824	0x5000c3fc	4	CR
12	837	0x5000c548	4	CR
13	837	0x5000c549	4	CR
14	857	0x5000c907	4	CR
15	1024	0x5000f513	4	CR
16	1032	0x5000ff20	4	CR

Table 2: Observations of CRL618 after OD189; observations marked in yellow were not made at the nominal sky position.

#	OD	Obsid	Number of repetitions	Spectral resolution
1	217	0x50002aa6	5	CR
2	227	0x50002c7d	5	CR
3	240	0x50002e42	5	CR
4	240	0x50002e43	17	HR
5	354	0x5000483a	5	CR
6	395	0x500051fe	5	CR
7	410	0x500055d3	5	CR
8	543	0x50007983	4	CR
9	572	0x50008328	4	CR
10	601	0x500088e5	4	CR
11	711	0x5000a532	4	CR
12	767	0x5000b218	4	CR
13	971	0x5000e950	4	CR

Table 3: Observations of AFGL2688 after OD189; observations marked in yellow were not made at the nominal sky position.

#	OD	Obsid	Number of repetitions	Spectral resolution
1	288	0x50003678	17	HR
2	1012	0x5000f281	4	CR
3	1024	0x5000f50e	4	CR

Table 4: Observation of NGC6302 after OD189.

#	OD	Obsid	Number of repetitions	Spectral resolution
1	240	0x50002E3B	17	HR
2	543	0x5000797C	4	CR
3	557	0x50007F3E	4	CR
4	602	0x500088F5	4	CR
5	625	0x50008D06	4	CR
6	654	0x5000929E	4	CR
7	742	0x5000AD8E	4	CR
8	767	0x5000B20D	4	CR
9	803	0x5000B96B	4	CR
10	824	0x5000C401	4	CR
11	837	0x5000C541	4	CR
12	856	0x5000C8DA	4	CR
13	886	0x5000D212	4	CR
14	971	0x5000e942	4	CR
15	988	0x5000ee20	4	CR
16	997	0x5000efe8	4	CR
17	1012	0x5000f282	4	CR

Table 5: Observations of AFGL4106 after OD189; observations marked in yellow were not made at the nominal sky position.

The observations of the individual sources were made close to the nominal sky positions. SPIRE's beam steering mirror put the actual sky position on a circle of about 1.7 arcsec radius around the nominal sky position until OD 987. The actual sky positions for the various observations are given below for each source:

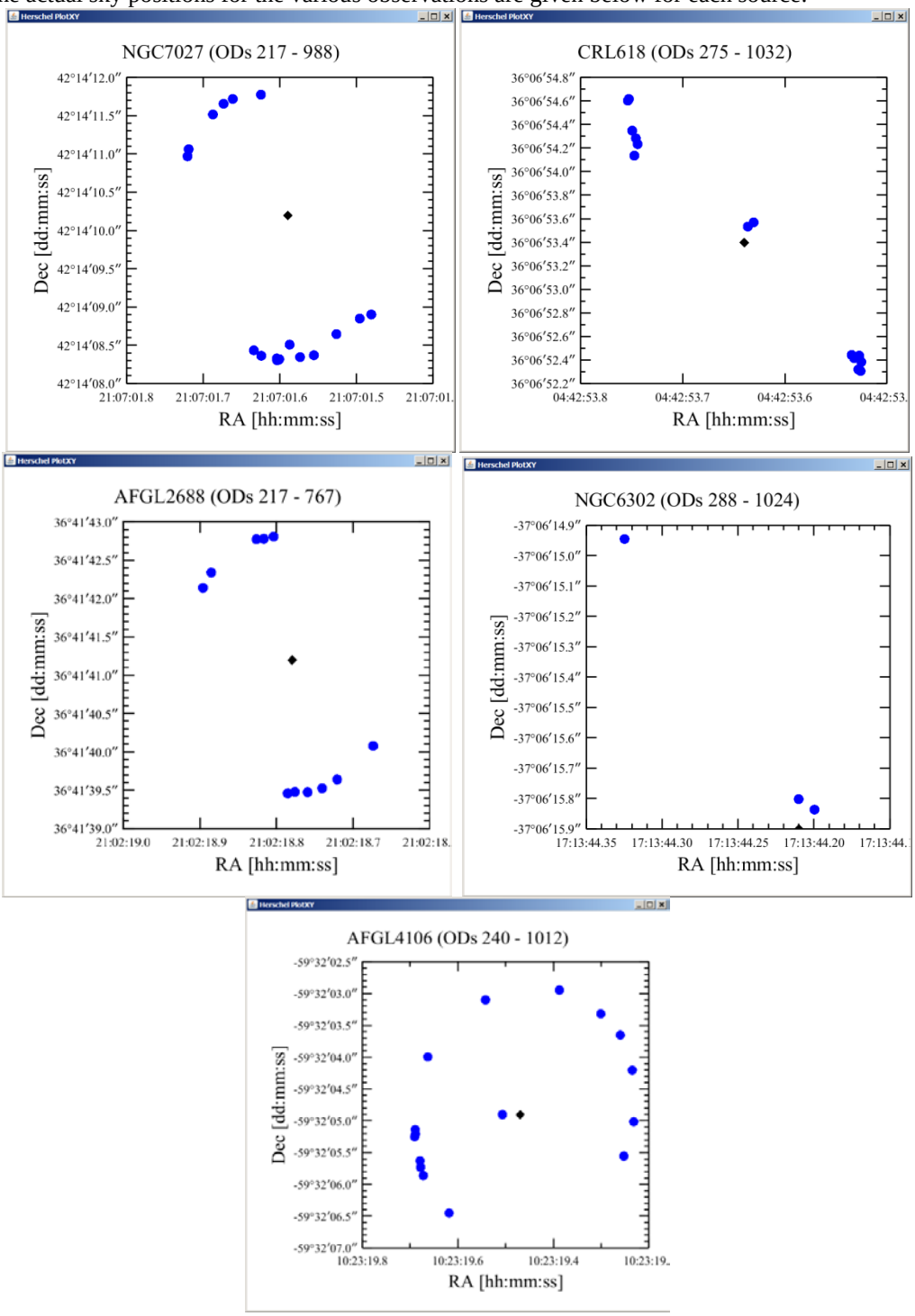


Figure 1: Actual sky positions (blue) for each source in relation to the nominal sky position (black)

3. DATA PROCESSING

All data were processed with the unaltered SOF1 SPG script from HIPE 9.0.1930 and the SPIRE Calibration context spire_cal_9_0.

No additional background subtraction was performed and the section below in yellow does not apply to the update of this document to issue 1.2.

The SPG products from the observation context were then processed further to clean up the continuum shape of the spectrum (particularly for SLW), allowing for an improved low-order, polynomial fit. The SPIRE Useful Script Spectrometer_BackgroundSubtraction.py was customized to operate on all relevant observations, more specifically the unapodized spectral products at level 1, and subtract the average sky background as measured by the co-aligned off-axis detectors. This step has the largest impact on NGC6302 where the off-axis detectors measured a signal of up to 10 Jy, see Figure 2.

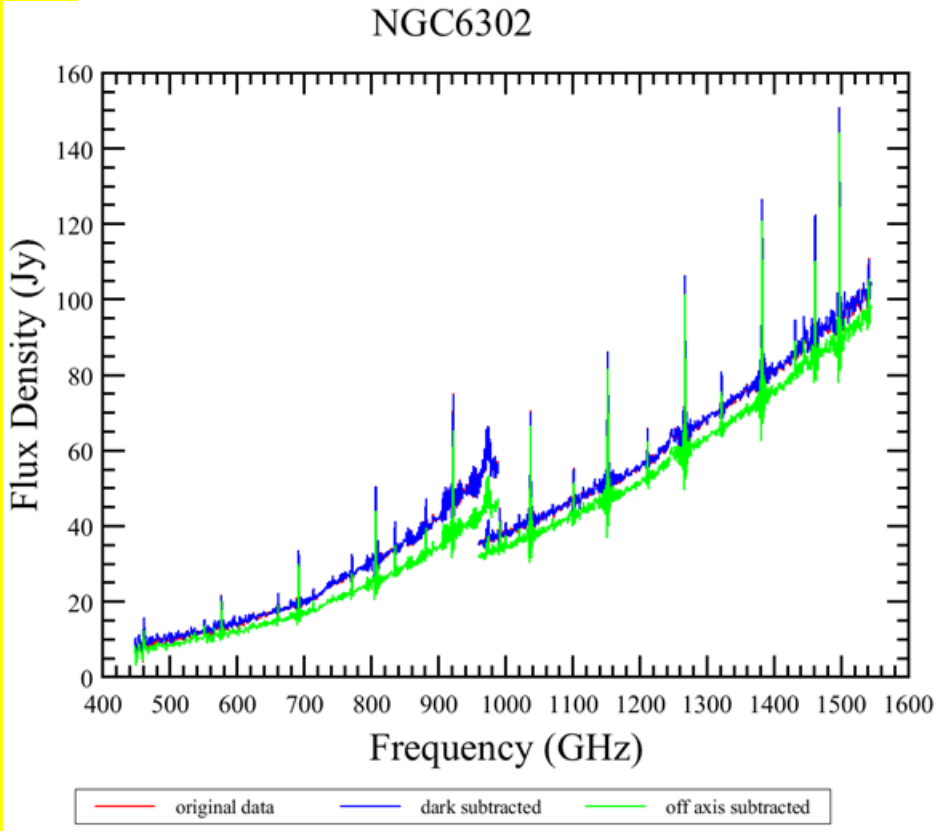


Figure 2: Signal from NGC6302 (obsid: 0x50003678) without (blue) and with background subtraction (green)

The script BulkOffAxisSubtraction.py is available via the Herschel Concurrent Versions System (CVS) and the version used for this work is reproduced in the Appendix.

4. DATA ANALYSIS

The SPIRE Useful Script Spectrometer_LineFitting.py was customized to analyze all spectra that were processed as described above. The script iterates through a list of sources and, for each source, a specific list of obsid's and

significant lines to be fitted. Only the results from the ^{12}CO transitions (see Table 6) are reported even though more lines are used to achieve the best fit.

Transition	SPIRE FTS band	Rest Frequency [GHz]
^{12}CO 4-3	SLW	461.0407682
^{12}CO 5-4	SLW	576.2679305
^{12}CO 6-5	SLW	691.4730763
^{12}CO 7-6	SLW	806.6518060
^{12}CO 8-7	SLW	921.7997000
^{12}CO 9-8	SSW	1036.9123930
^{12}CO 10-9	SSW	1151.9854520
^{12}CO 11-10	SSW	1267.0144860
^{12}CO 12-11	SSW	1381.9951050
^{12}CO 13-12	SSW	1496.9229090

Table 6: Assumed rest frequencies for the ^{12}CO transitions within the SPIRE band.

When processing a specific observation, the radial velocity of the spacecraft with respect to the Local Standard of Rest (LSR), as reported in the metadata of the observation context in the Herschel Science Archvie (HSA), is used to correct the rest frequencies of the ^{12}CO transitions given in Table 6:

frequencyCorrected = frequency * (1 - radialVelocity/c)

When fitting, the source is assumed to be at rest with respect to the LSR. The derived velocities represent the overall source velocity with respect to the LSR. The script uses the Levenberg-Marquardt fitter to match the spectra to a model that adds a set of sinc functions to a third order polynomial function (for the continuum). Data from the two central detectors of the SPIRE FTS (SSWD4 and SLWC3) are analyzed separately. Each sinc model has three parameters. Two parameters are kept free, the frequency and the amplitude of the peak, whereas the width of the sinc function is set to the value of the metadatum "actualResolution" – nominally 1.1854 GHz – which represents the width of the spectral resolution element from the SPIRE FTS, divided by π. Initial guesses for the free parameters are zeros for all polynomial coefficients and, for the sinc functions, the corrected rest frequency and the flux value of the measured spectrum at the corrected rest frequency are the initial guesses for the sinc functions. The script fits to all significant lines in a spectrum and reports results for the ^{12}CO transitions. For each sinc model, the integrated line flux and the derived source velocity are calculated as below:

Line flux [W m^{-2}] = 10^{-26} * peakAmplitude [Jy] * 10^{9} * π * actualResolution [GHz]
Velocity [km s^{-1}] = c [km s^{-1}] * (frequencyCorrected [GHz] – peakFrequency [GHz]) / frequencyCorrected [GHz]

Error estimates can be calculated based on the standard deviation reported by the fitter. The line flux and velocity errors are calculated as follows:

Line flux error [W m^{-2}] = Line flux [W m^{-2}] * peakAmplitudeStdDev [Jy] / peakAmplitude [Jy]
Velocity error [km s^{-1}] = c [km s^{-1}] * peakFrequencyStdDev [GHz] / frequencyCorrected [GHz]

Important note on reported standard deviations: Due to a problem when performing a constrained fit in HIPE, the reported standard deviation values from the fitter are not necessarily meaningful, see HCSS-14964. Instead of reporting potentially misleading values from the fitter, the standard deviation of the set of values for all observations of the same source is reported. Divide this number by the square root of the number of observations to get an indication of the uncertainty of the reported average values.

The script also creates one table dataset with line flux and velocity values per observation and stores it as a fits file (SpireResultsLineSources.fits). The fits file contains:

- Column "rest frequencies": Nominal ^{12}CO line frequencies in units of GHz.
- Column "Line Flux <obsid>": integrated line flux for the specified observation for each transition in units of W m^{-2}.
- Column "Line Velocity <obsid>": line velocity for the specified observation for each transition in units of km s^{-1}.

The script SpireAnalysisLineSources.py is available via the Herschel CVS and the version used for this work is reproduced in section 6.1 SpireAnalysisLineSources.py.

5. RESULTS

The observations at non-nominal sky positions (see yellow highlights in Table 1 to Table 5) were not used to derive the values reported in the following sections. The average and standard deviation values and plots can be recreated from the data contained in the table dataset stored in SpireResultsLineSources.fits.

5.1 Overview

The average values and the standard deviation of the routine calibration observations are given for each source. The number in brackets next to the source name indicates the number of analyzed observations.

Colored shading of values indicates that reported values need special consideration:

- Yellow: The source is slightly extended in comparison to Uranus, which defines the size of a point source. No attempt was made to account for the source extension.
- Blue: Close-by, non-^{12}CO lines affect the flux derived for ^{12}CO lines. No attempt was made to fit more than one sinc function to any given, potentially blended line feature. NB: Contamination for AFGL4106 has not been checked.
- Red: The point flux calibration is noisiest in the low frequency range (flux calibrator Uranus has the lowest flux at low frequencies) and the uncertainty in the derived velocity will be high.

	NGC7027 (18)		CRL618 (14)		AFGL2688 (11)		NGC6302 (3)		AFGL4106 (16)	
^{12}CO	Avg	Std Dev	Avg	Std Dev	Avg	Std Dev	Avg	Std Dev	Avg	Std Dev
4-3	567.0	12.1	179.0	16.6	292.0	30.2	62.2	8.4	68.2	20.9
5-4	826.9	23.5	294.1	22.2	470.3	36.6	96.9	15.3	134.4	17.6
6-5	1139.9	23.7	473.6	12.5	819.4	65.9	144.0	7.6	181.3	6.3
7-6	1664.0	25.4	735.7	14.2	1287.0	60.6	231.7	3.8	251.9	4.9
8-7	2197.0	28.3	1100.4	10.8	1936.0	36.1	351.3	2.4	291.0	7.4
9-8	1949.5	59.1	1405.6	52.5	2073.0	192.6	320.5	31.8	282.9	11.4
10-9	2079.6	79.1	2197.6	87.3	2441.3	901.0	386.2	39.5	272.4	11.7
11-10	2150.0	84.1	2276.1	89.5	2974.8	340.1	458.9	44.8	242.2	12.3
12-11	2109.7	102.8	2646.6	131.0	3250.2	381.7	506.7	60.6	197.5	9.8
13-12	2041.0	82.7	3071.9	119.6	3472.5	386.3	587.5	53.2	168.3	10.2

Table 7: Average and Standard Deviation of Integrated Line Flux [1E-18 W m^{-2}]

	NGC7027 (18)		CRL618 (14)		AFGL2688 (11)		NGC6302 (3)		AFGL4106 (16)	
^{12}CO	Avg	Std Dev	Avg	Std Dev	Avg	Std Dev	Avg	Std Dev	Avg	Std Dev
4-3	37.3	15.8	20.7	73.1	-22.0	24.3	119.0	142.4	2.5	99.4
5-4	-3.1	4.0	-56.1	15.0	-50.3	10.3	-75.9	57.7	-31.2	36.0
6-5	30.9	3.2	4.5	4.9	6.0	4.0	-22.2	10.2	-9.7	7.8
7-6	23.5	3.9	-29.7	4.9	-43.7	4.7	-47.3	4.3	-23.7	6.4
8-7	30.0	5.1	-19.3	5.6	-36.0	7.6	-34.0	4.9	-13.8	7.9
9-8	23.9	2.6	-25.1	4.9	-38.4	2.4	-39.1	3.3	-20.7	4.4
10-9	29.3	3.0	-0.2	6.0	25.4	34.7	-38.0	3.5	-17.2	4.3
11-10	25.1	3.9	-31.0	5.7	-39.9	4.0	-38.4	2.5	-16.8	5.5
12-11	25.2	4.5	-26.2	6.4	-39.7	5.0	-41.2	2.7	-15.9	7.5
13-12	20.4	5.5	-29.3	6.5	-40.6	7.9	-44.3	3.2	-23.4	9.7

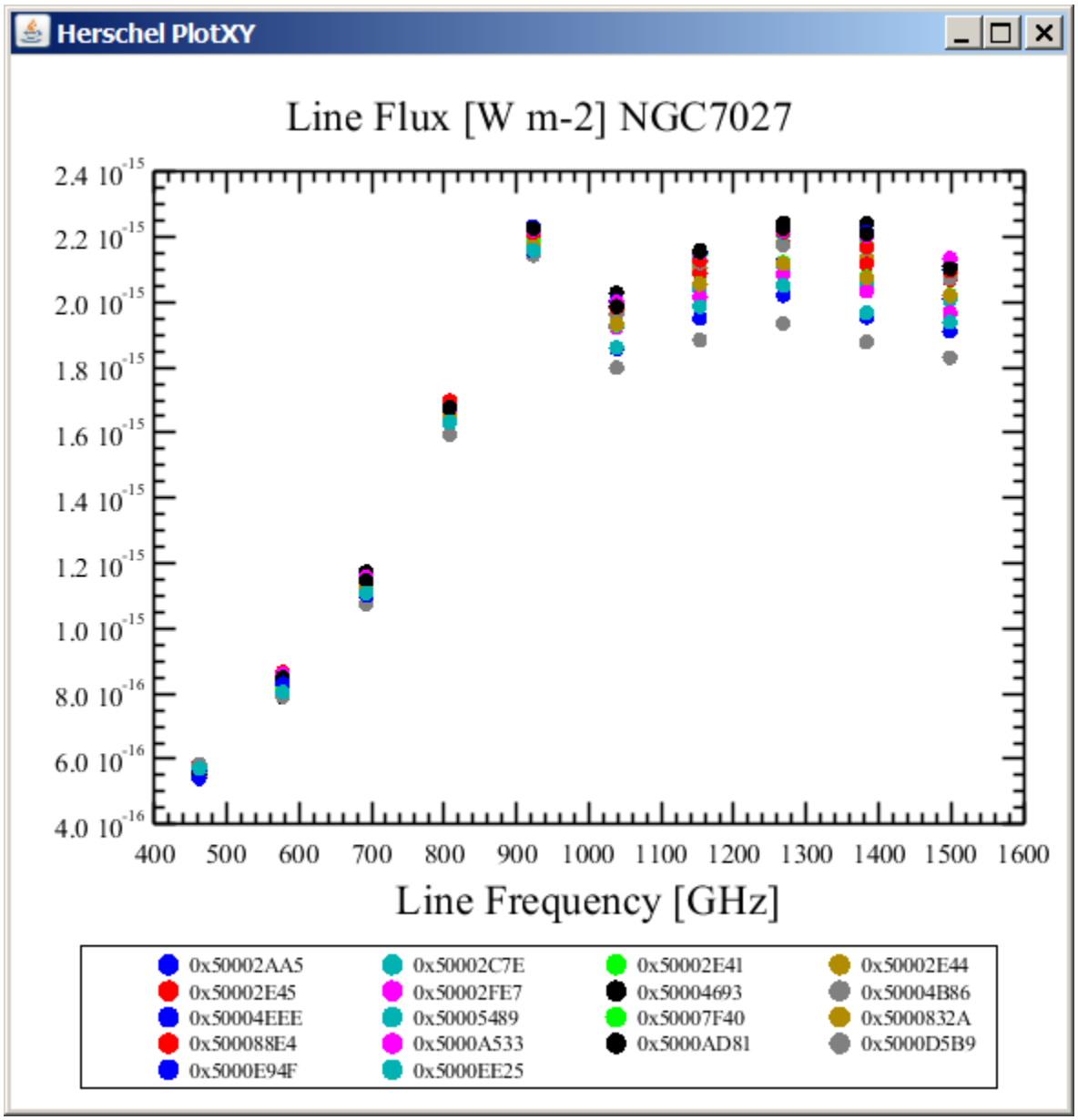
Table 8: Average and Standard Deviation of Line Velocities [km s^{-1}]

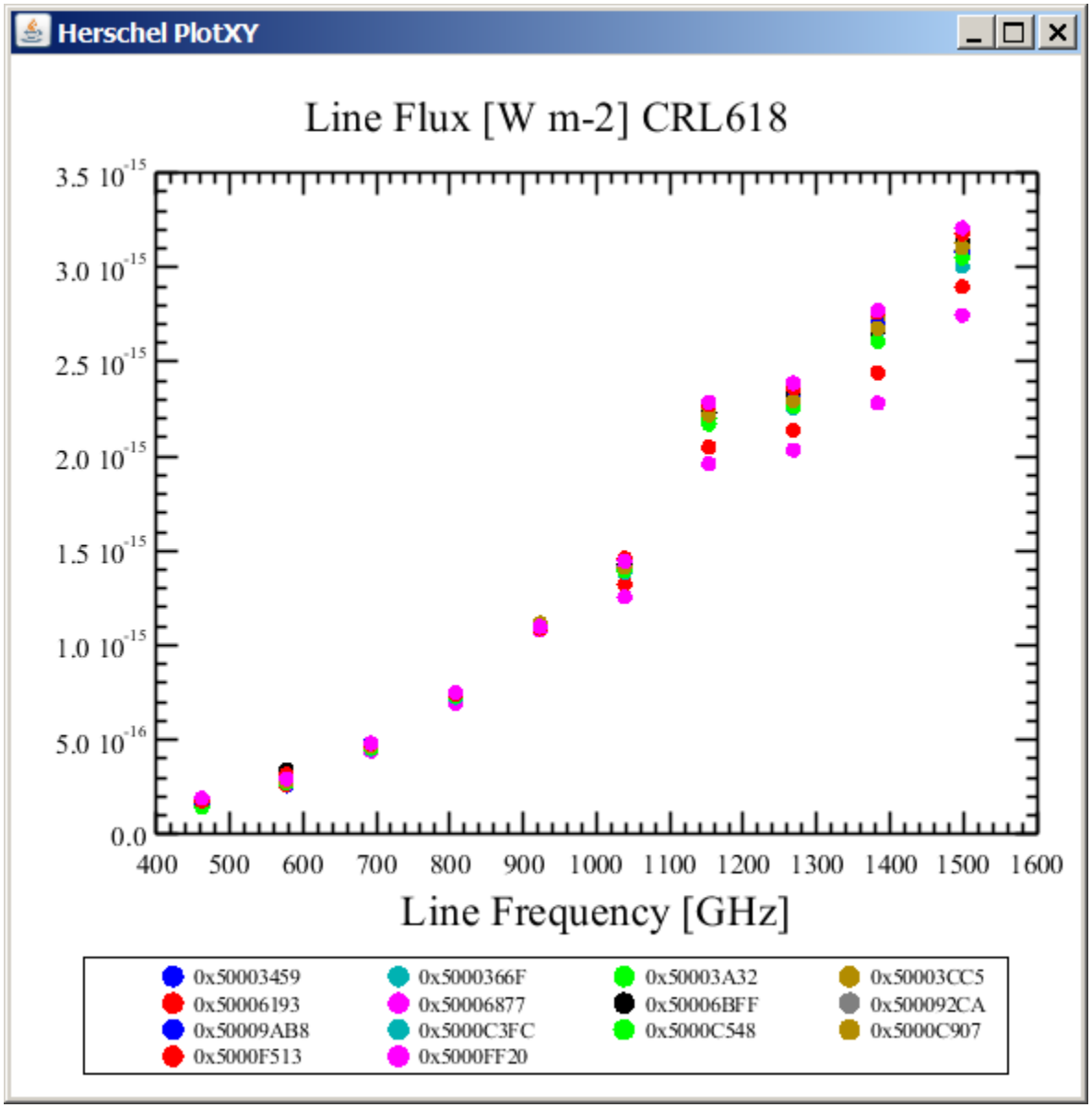
The source velocities and their uncertainty estimates are derived from the non-shaded entries in Table 8:

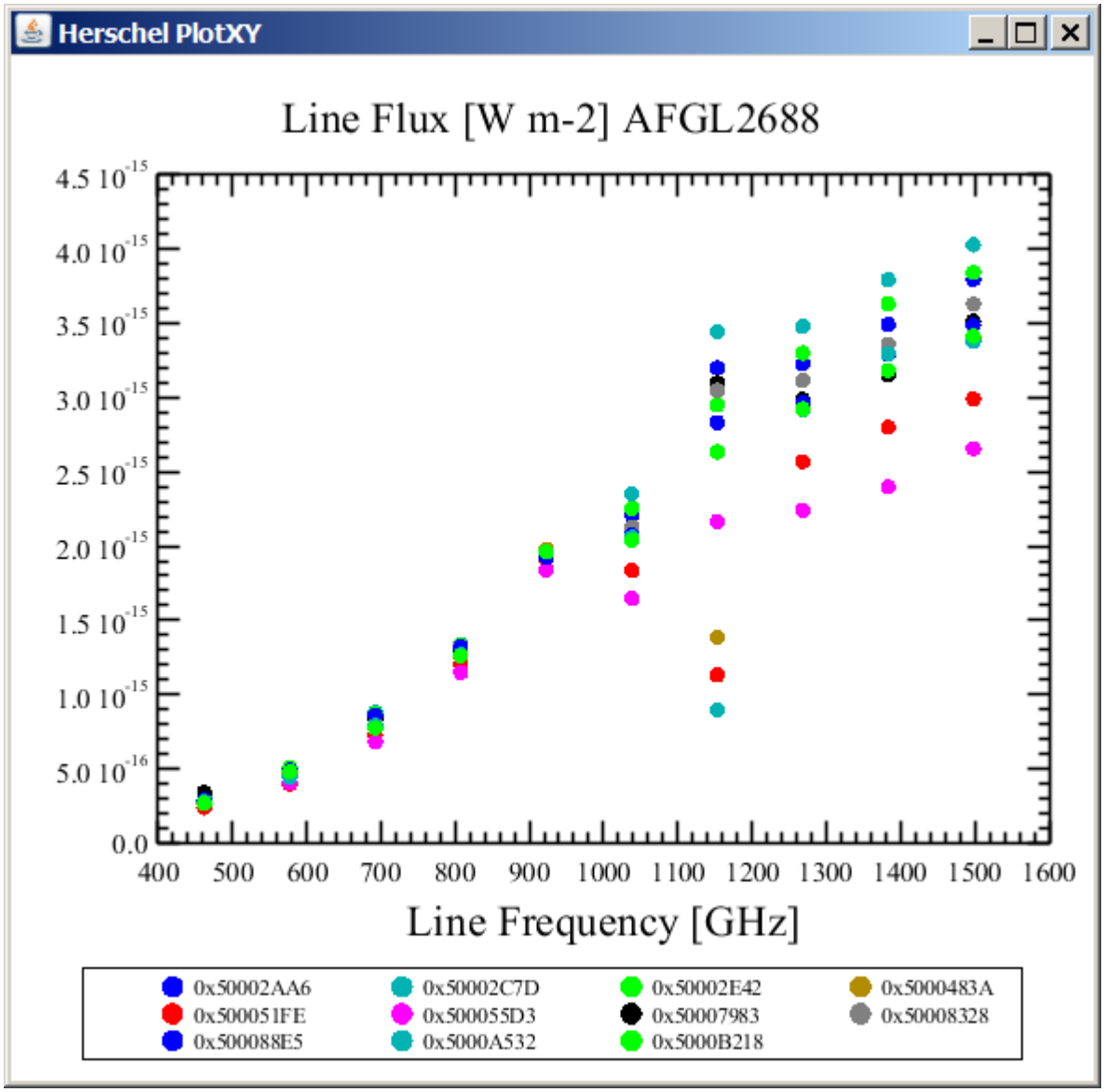
Source	*Average velocity [km s^{-1}]*	*Average standard deviation [km s^{-1}]*
NGC7027	26.0	4.0
CRL618	-26.8	5.7
AFGL2688	-39.7	5.3
NGC6302	-38.0	4.3
AFGL4106	-17.7	6.7

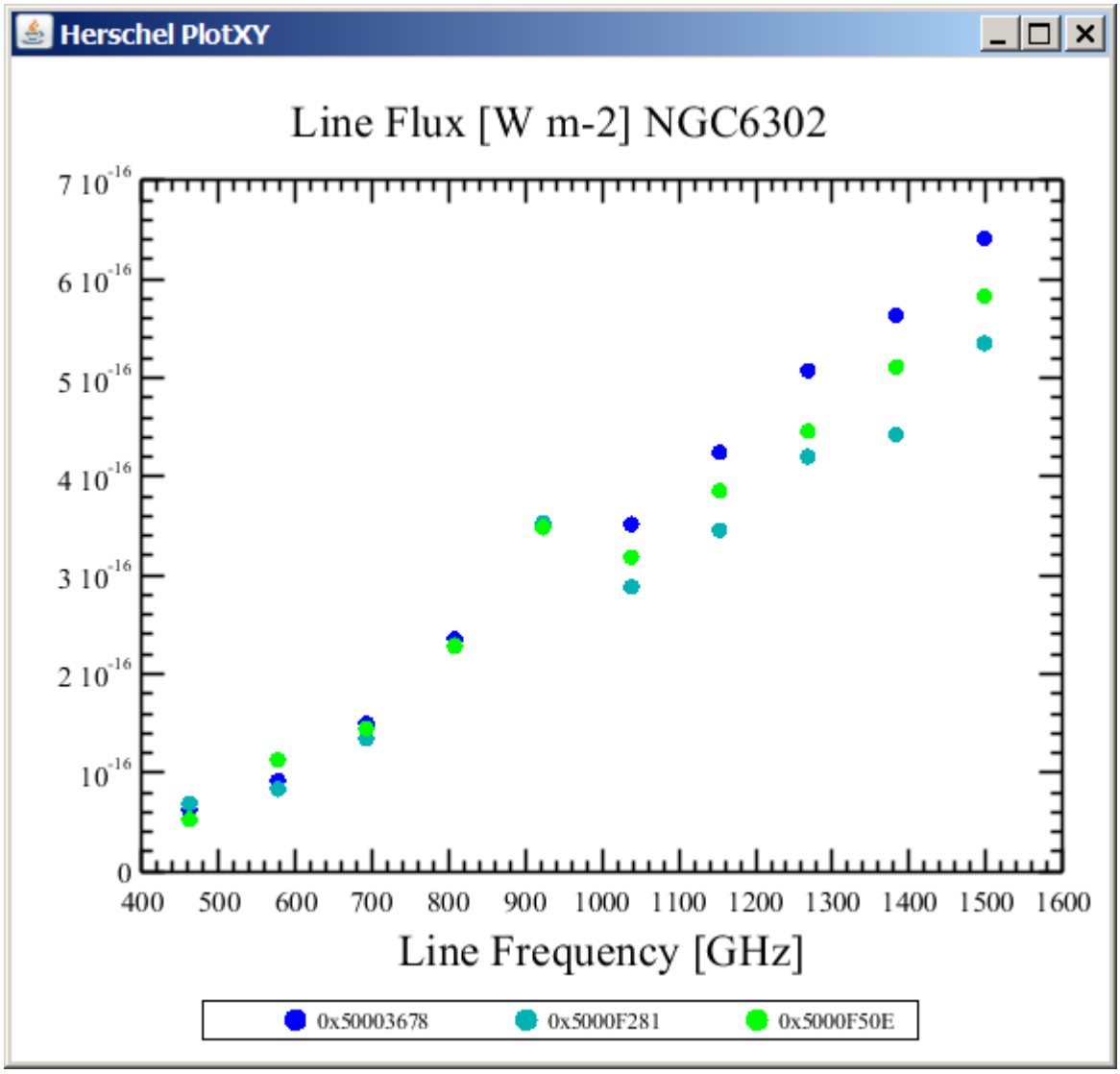
Table 9: Derived source velocities and their uncertainties

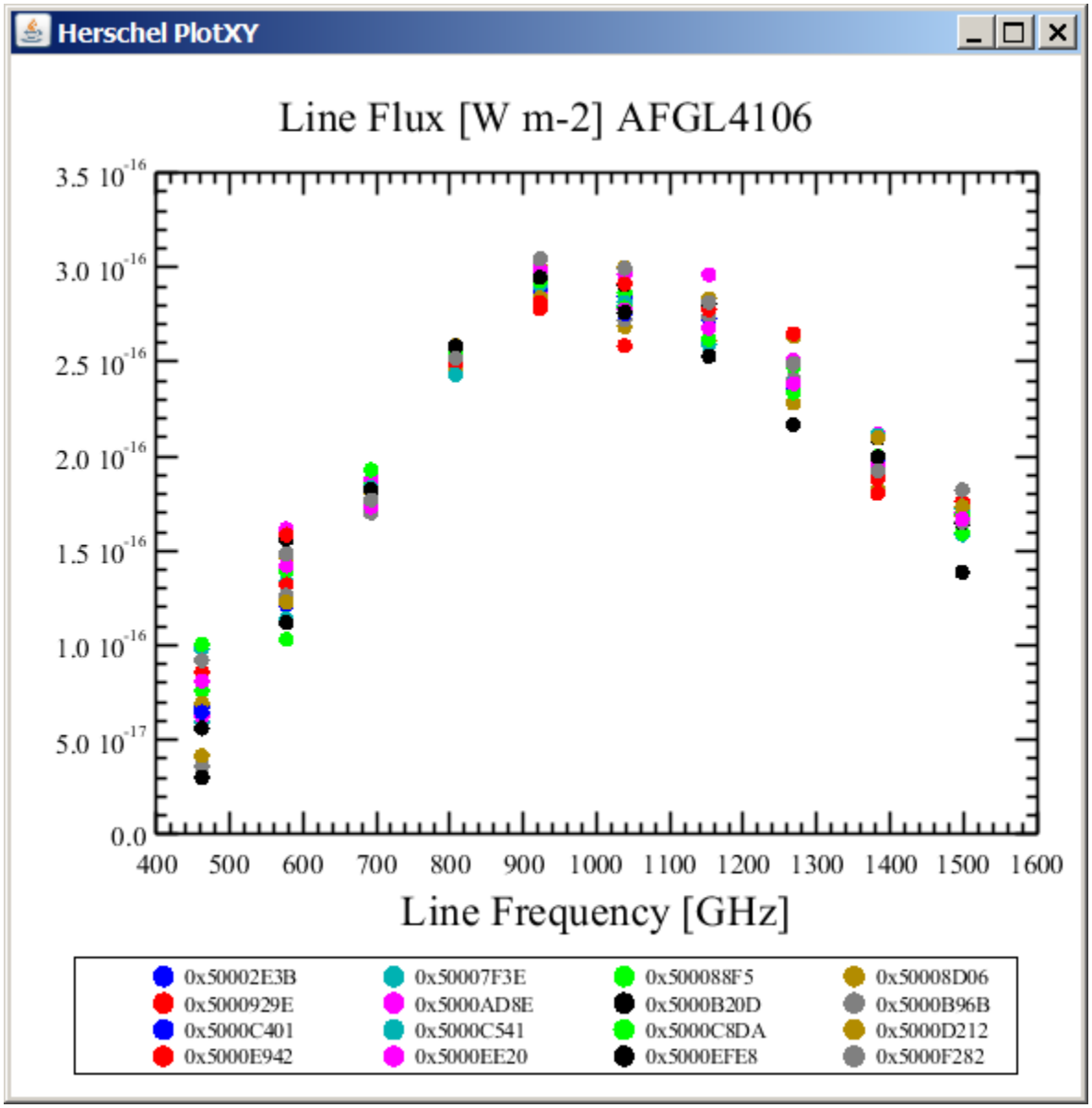
5.2 Line Flux values per observation



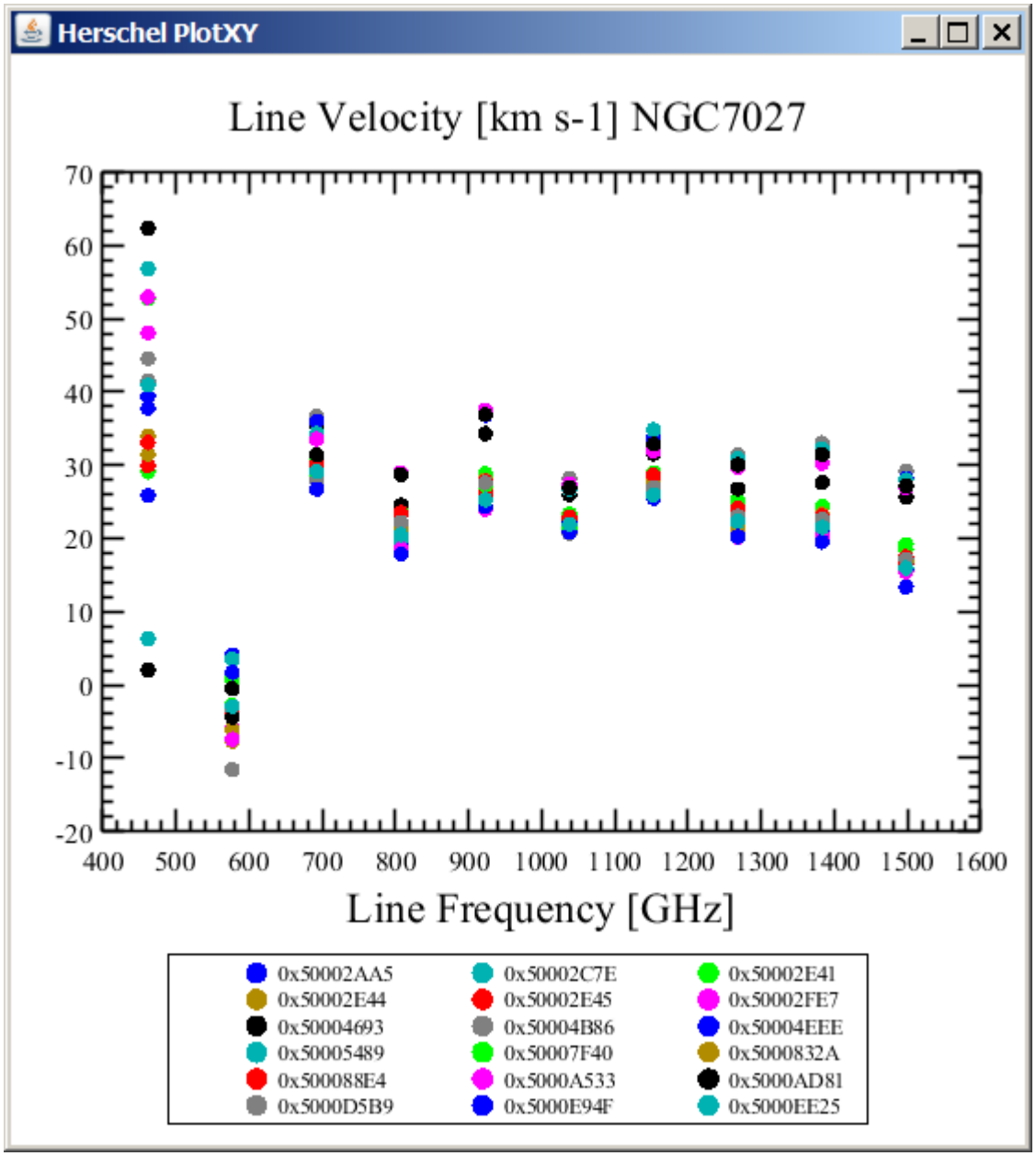

Herschel PlotXY
Line Flux [W m-2] CRL618
3.5 10-15
3.0 10-15
2.5 10-15
2.0 10-15
1.5 10-15
1.0 10-15
5.0 10-16
0.0
400 500 600 700 800 900 1000 1100 1200 1300 1400 1500 1600
Line Frequency [GHz]
0x50003459
0x50006193
0x50009AB8
0x5000F513
0x5000366F
0x50006877
0x5000C3FC
0x5000FF20
0x50003A32
0x50006BFF
0x5000C548
0x50003CC5
0x500092CA
0x5000C907

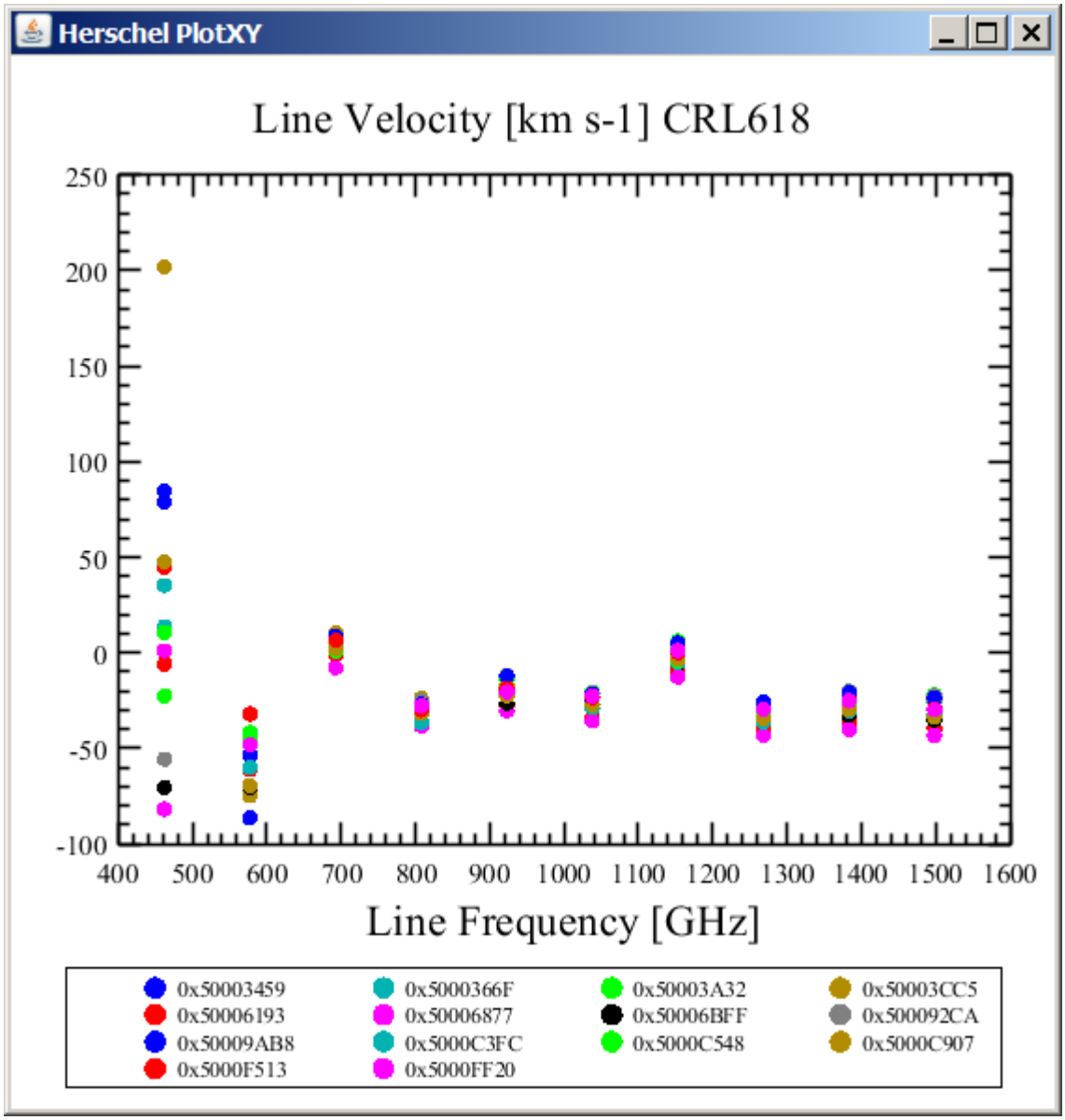

Herschel PlotXY
Line Flux [W m-2] AFGL2688
4.5 10-15
4.0 10-15
3.5 10-15
3.0 10-15
2.5 10-15
2.0 10-15
1.5 10-15
1.0 10-15
5.0 10-16
0.0
400 500 600 700 800 900 1000 1100 1200 1300 1400 1500 1600
Line Frequency [GHz]
0x50002AA6
0x50002C7D
0x50002E42
0x5000483A
0x500051FE
0x500055D3
0x50007983
0x50008328
0x500088E5
0x5000A532
0x5000B218

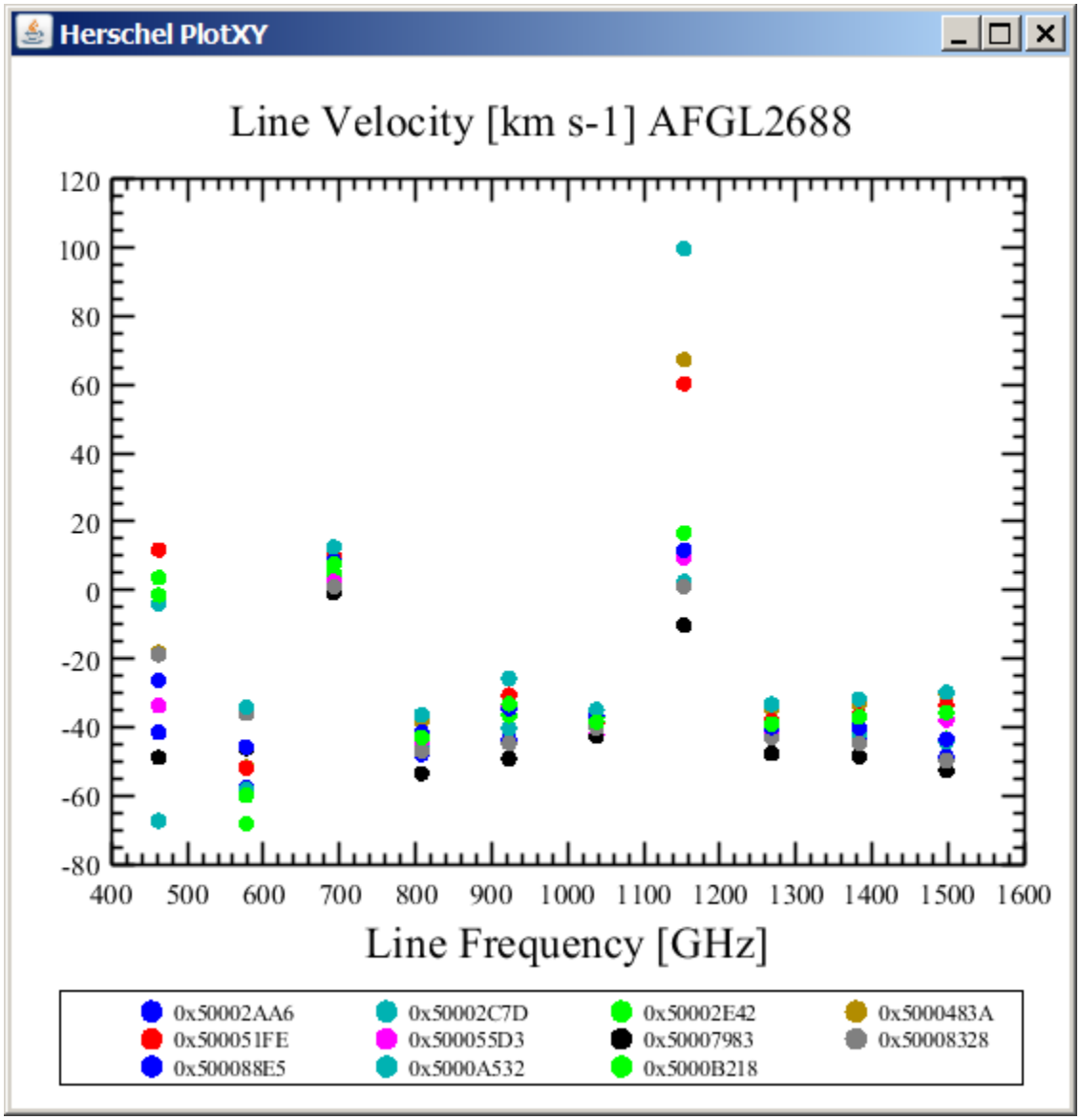

Herschel PlotXY
Line Flux [W m-2] NGC6302
7 10-16
6 10-16
5 10-16
4 10-16
3 10-16
2 10-16
10-16
0
400
500
600
700
800
900
1000
1100
1200
1300
1400
1500
1600
Line Frequency [GHz]
0x50003678
0x5000F281
0x5000F50E

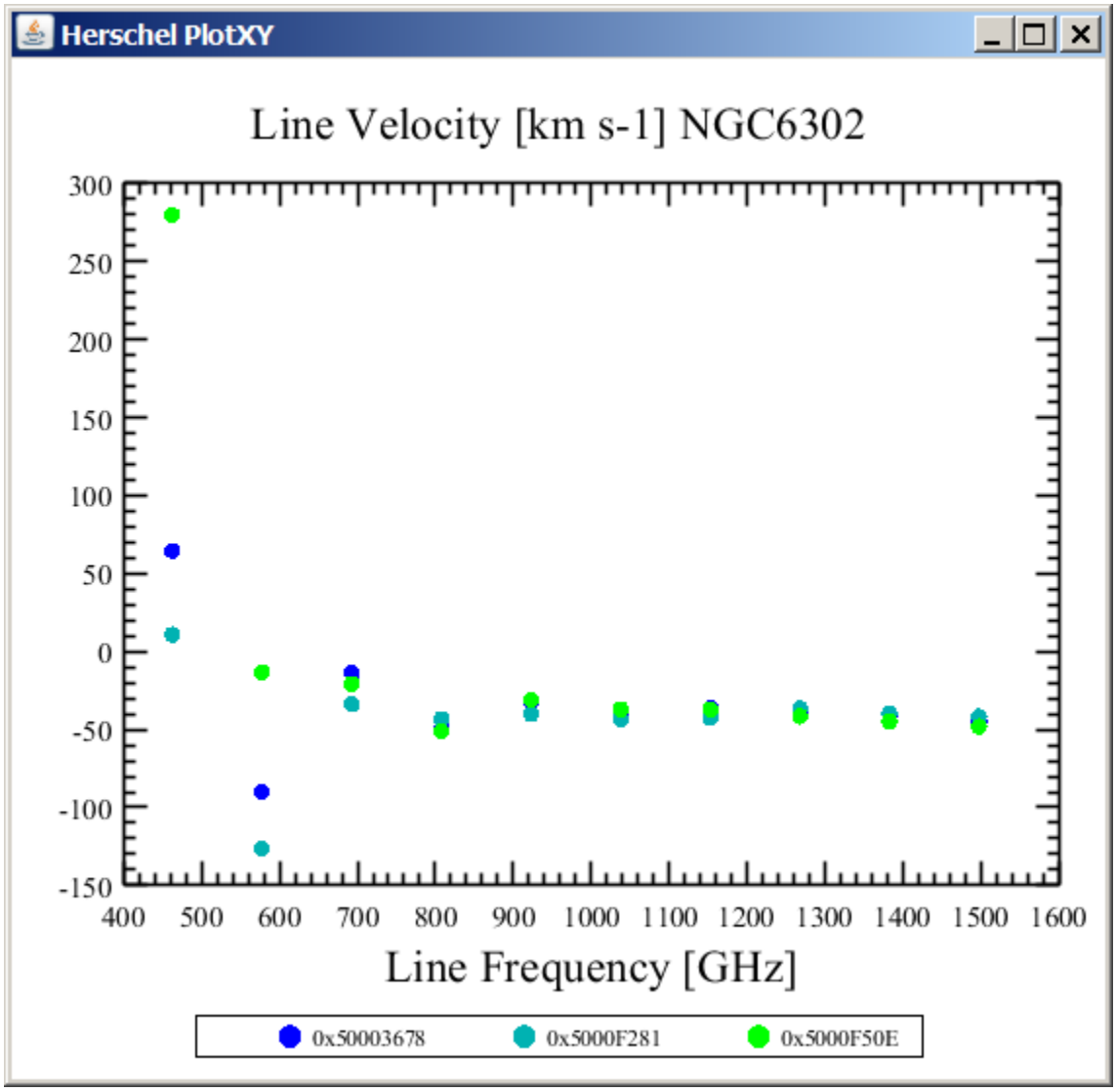

Herschel PlotXY
Line Flux [W m-2] AFGL4106
3.5 10-16
3.0 10-16
2.5 10-16
2.0 10-16
1.5 10-16
1.0 10-16
5.0 10-17
0.0
400 500 600 700 800 900 1000 1100 1200 1300 1400 1500 1600
Line Frequency [GHz]
0x50002E3B
0x5000929E
0x5000C401
0x5000E942
0x50007F3E
0x5000AD8E
0x5000C541
0x5000EE20
0x500088F5
0x5000B20D
0x5000C8DA
0x5000EFE8
0x50008D06
0x5000B96B
0x5000D212
0x5000F282

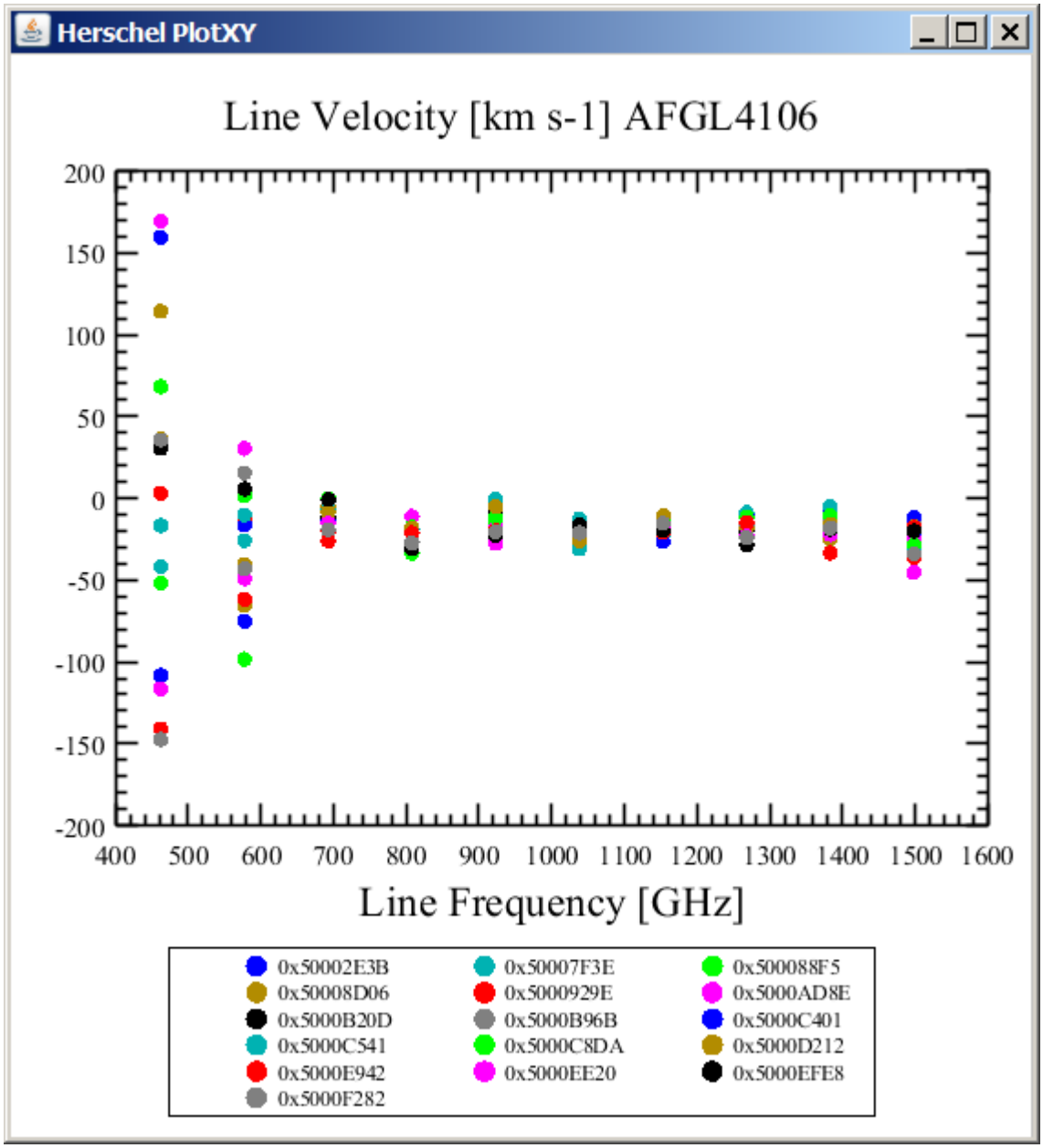
5.3 Line Velocity values per observation




Herschel PlotXY
Line Velocity [km s-1] CRL618
250
200
150
100
50
0
-50
-100
400
500
600
700
800
900
1000
1100
1200
1300
1400
1500
1600
Line Frequency [GHz]
0x50003459
0x50006193
0x50009AB8
0x5000F513
0x5000366F
0x50006877
0x5000C3FC
0x5000FF20
0x50003A32
0x50006BFF
0x5000C548
0x50003CC5
0x500092CA
0x5000C907


Herschel PlotXY
Line Velocity [km s-1] AFGL2688
120
100
80
60
40
20
0
-20
-40
-60
-80
400
500
600
700
800
900
1000
1100
1200
1300
1400
1500
1600
Line Frequency [GHz]
0x50002AA6
0x500051FE
0x500088E5
0x50002C7D
0x500055D3
0x5000A532
0x50002E42
0x50007983
0x5000B218
0x5000483A
0x50008328


Herschel PlotXY
Line Velocity [km s-1] NGC6302
300
250
200
150
100
50
0
-50
-100
-150
400
500
600
700
800
900
1000
1100
1200
1300
1400
1500
1600
Line Frequency [GHz]
0x50003678
0x5000F281
0x5000F50E


Herschel PlotXY
Line Velocity [km s-1] AFGL4106
200
150
100
50
0
-50
-100
-150
-200
400
500
600
700
800
900
1000
1100
1200
1300
1400
1500
1600
Line Frequency [GHz]
0x50002E3B
0x50007F3E
0x500088F5
0x50008D06
0x5000929E
0x5000AD8E
0x5000B20D
0x5000B96B
0x5000C401
0x5000C541
0x5000C8DA
0x5000D212
0x5000E942
0x5000EE20
0x5000EFE8
0x5000F282

6. APPENDIX

6.1 SpireAnalysisLineSources.py

SpireAnalysisLineSources.py from CVS\develop\user\ca\lethbridge\pdavis\Scripts\XCAL\

```
"""
<!--
$Id: SpireAnalysisLineSources.py,v 1.1 2012/05/22 21:50:31 pdavis Exp $
-->
"""

# This script fits the continuum and line features for a set of line point
# sources to characterize line features.
#
# It can operate on many sources, many obsid's, many V&V versions and flavors.
# This version of the script is designed for routine and cross-calibration.
#
# Line features include integrated line flux and source velocity. Each source
# is also analyzed for the achieved line sensitivity.
#
# The script assumes that SpectrometerDetectorSpectrum products (from the V&V)
# are available on the hard-drive.
#
# The script can create the following plots:
# 1) plotSkyPositions - For each source: actual sky positions per observation
# 2) plotData - For each source: average and standard deviation of line flux and velocity and
sensitivity
# 3) plotObs - For each source: line flux and velocity per observation
# 4) plotFit - For each measured spectrum: fitting information from SpectrumFitter (for debugging
the fit)

############################
#     Plot Selections      #
#                          #
#1)                        #
plotSkyPositions = False
#                          #
#2)                        #
plotData = False
printData = True
#                          #
#3)                        #
plotObs = True
#                          #
#4)                        #
plotFit = False
#                          #
############################

# Define constants
c = herschel.share.unit.Constant.SPEED_OF_LIGHT.value
PI = Math.PI
resWinSize = 25.0 # [GHz] window size to calculate residual
CO12Lines = Double1d([461.0407682,576.2679305,691.4730763,806.6518060,921.7997000,\
                  1036.9123930,1151.9854520,1267.0144860,1381.9951050,1496.9229090])
path = '/SPIRE/V_and_V/'      # data location

# Prepare the dataset to contain all information
SpireResultsLineSources = TableDataset()
SpireResultsLineSources['Rest
Frequencies']=Column(CO12Lines,unit=herschel.share.unit.Frequency.GIGAHERTZ,description='12CO
Rest Frequencies')
```

```
# ['CRL618','NGC7027','AFGL2688','NGC6302','AFGL4106','AFGL2591']
sources = ['CRL618','NGC7027','AFGL2688','NGC6302','AFGL4106','AFGL2591']
# Define data processing version and flavor
calibrations = ['calibration_tree']
versions = ['HCSS_9_0_0_ALPHA']

# Define source name, OD range, observation IDs, line positios, line count
#
for src in sources:
  print src
  if plotObs:
    pFlux = PlotXY(titleText='Line Flux [W m-2] %s'%src)
    pVel = PlotXY(titleText='Line Velocity [km s-1] %s'%src)
  if src == 'CRL618':
    days = '275 - 1032'
    obsids = Double1d([0x50003459, \
    #0x5000366E, \ not at the nominal sky position
    0x5000366F, \
    0x50003A32, \
    0x50003CC5, \
    0x50006193, \
    0x50006877, \
    0x50006BFF, \
    0x500092CA, \
    0x50009AB8, \
    0x5000C3FC, \
    0x5000C548, \
    0x5000C907, \
    0x5000F513, \
    0x5000FF20, \
    ]) #
    # include any lines to be verified
    lineFreqs = Double1d(CO12Lines)
    lineFreqs.append(Double1d([532,620,709,797.5,886,\
                           ]))
    lineFreqs = SORT(lineFreqs)
  elif src == 'NGC7027':
    days = '217 - 988'
    obsids = Double1d([ \
    0x50002AA5, \
    0x50002C7E, \
    0x50002E41, \
    0x50002E44, \
    0x50002E45, \
    0x50002FE7, \
    0x50004693, \
    0x50004B86, \
    0x50004EEE, \
    0x50005489, \
    0x50007F40, \
    0x5000832A, \
    0x500088E4, \
    0x5000A533, \
    0x5000AD81, \
    0x5000D5B9, \
    0x5000E94F, \
    0x5000EE25, \
    ])
    # 1342193812  see http://herschel.esac.esa.int/twiki/bin/view/Spire/SpireSpecObsKnownIssues
    # include any lines to be verified
    lineFreqs = Double1d(CO12Lines)
    lineFreqs.append(Double1d([835.07189\
                      ]))
    lineFreqs = SORT(lineFreqs)
  elif src == 'AFGL2688':
```

```
  days = '217 - 971'
  obsids = Double1d([0x50002AA6, \
  0x50002C7D, \
  #0x50002E43, \ Unusual sky position - about 5" from nominal
  0x50002E42, \
  0x5000483A, \
  0x500051FE, \
  0x500055D3, \
  0x50007983, \
  0x50008328, \
  0x500088E5, \
  0x5000A532, \
  0x5000B218, \
  #0x5000E950, \
  ])
  # include any lines to be verified
  lineFreqs = Double1d(CO12Lines)
  lineFreqs.append(Double1d([532,551,620,661,709,771,797,881,886,\
                   1063,1101,1153,1240,1328,1417,1505]))
  lineFreqs = SORT(lineFreqs)
elif src == 'AFGL4106':
  days = '240 - 1012'
  obsids = Double1d([0x50002E3B, \
  # 0x5000797C, \ see http://herschel.esac.esa.int/twiki/bin/view/Spire/SpireSpecObsKnownIssues
  0x50007F3E, \
  0x500088F5, \
  0x50008D06, \
  0x5000929E, \
  0x5000AD8E, \
  0x5000B20D, \
  0x5000B96B, \
  0x5000C401, \
  0x5000C541, \
  0x5000C8DA, \
  0x5000D212, \
  0x5000E942, \
  0x5000EE20, \
  0x5000EFE8, \
  0x5000F282, \
  ])
  # include any lines to be verified
  lineFreqs = Double1d(CO12Lines)
  lineFreqs.append(Double1d([532,661,770.5,881,\
                   1461]))
  lineFreqs = SORT(lineFreqs)
elif src == 'NGC6302':
  days = '288 - 1024'
  obsids = Double1d([0x50003678, \
  0x5000F281, \
  0x5000F50E, \
  ])
  # include any lines to be verified
  lineFreqs = Double1d(CO12Lines)
  lineFreqs.append(Double1d([532,551,620,661,709,771,797,835,881,886,\
                   1101,1211.5,1321.5,1431,1460.5,1505]))
  lineFreqs = SORT(lineFreqs)
elif src == 'AFGL2591':
  days = '544 - 682'
  obsids = Double1d([0x50007CFF, \
                     0x50009ABC])
  # include any lines to be verified
  lineFreqs = Double1d(CO12Lines)
  lineFreqs.append(Double1d([532,551,620,661,709,771,797,835,881,886,\
                   1101,1211.5,1321.5,1431,1460.5,1505]))
  lineFreqs = SORT(lineFreqs)
pass
```

```
  try:
    foo = asciiTableReader('C:\\SPIRE\\TMP\\%s_velocities.txt'%src)
    velocities = foo['velocities'].data
    if len(velocities) <> len(obsids): stop
    print 'Read s/c velocities from file'
  except:
    # archive access to retrieve radialVelocity
    print 'Access s/c velocities from HSA'
    archive = HsaReadPool()
    hsa_storage = ProductStorage()
    hsa_storage.register(archive)
    velocities = Double1d()
    for obsid in obsids:
        # retrieve radial velocity from HSA
        queryStr = "p.obsid == '%i'"%(obsid)
        query = MetaQuery(herschel.ia.obs.ObservationContext, 'p', queryStr)
        product = hsa_storage.select(query)
        for key in range(len(product)):
          obs = product[key].product
          radialVelocity = obs.meta["radialVelocity"].value
          velocities.append(radialVelocity)
        pass
    pass
    tds = TableDataset(description='radial velocity for %s observations'%src)
    tds['velocities']                          =                          Column(data=velocities,
unit=herschel.share.unit.Speed.KILOMETERS_PER_SECOND )
    asciiTableWriter(table=tds, file='C:\\SPIRE\\TMP\\%s_velocities.txt'%src)
  pass
  # Correct the frequencies for the source velocity (for initial guess)
  sourceVel = 0.0*1000.0
  lineFreqsCorr = lineFreqs*(1.-sourceVel/c)
  #
  LineFlux = Double4d(len(calibrations),len(versions),len(CO12Lines),len(obsids))
  LineFluxError = Double4d(len(calibrations),len(versions),len(CO12Lines),len(obsids))
  LineVelocity = Double4d(len(calibrations),len(versions),len(CO12Lines),len(obsids))
  LineVelocityError = Double4d(len(calibrations),len(versions),len(CO12Lines),len(obsids))
  LineSensitivity = Double4d(len(calibrations),len(versions),len(CO12Lines),len(obsids))
  #
  dts = Double1d(len(obsids))
  #
  SkyPosition_RA = Double1d()
  SkyPosition_DEC = Double1d()
  #
  j = 0
  for calibration in calibrations:
   i = 0
   print calibration
   for version in versions:
    print version
    obsCnt = 0
    for obsid in obsids:
      #print hex(int(obsid))
      lineCnt = 0
      myIndex = obsids.where(obsids == obsid)
      radialVelocity = 1000*velocities[myIndex][0]
      # Get the SDS containing the fully processed spectrum
      try:
       filename                                                                                                    =
'%s%s/level_2/%s/POINTSOURCE_FLUX_CONVERTED_ASDS_0x%8.8X_0xA1060001_Point_0_Jiggle_0_unapod_HR.fi
ts'%(path,version,calibration,obsid)
       sds = FitsArchive().load(filename)
      except:
       print 'Could not read %s %s %s'%(filename, version, calibration)
       stop
      #
      SkyPosition_RA.append(sds[0]['SSWD4'].meta['ra'].value)
```

```
SkyPosition_DEC.append(sds[0]['SSWD4'].meta['dec'].value)
#
dt = sds.endDate.subtract(sds.startDate)/(3600e+6)
dts.append(dt)
# Loop over the two centre detectors
for detector in ["SLWC3","SSWD4"]:
  # Read the detector spectrum and convert to GHz
  spc = sds['0000'][detector].copy()
  if '8' in version: spc = convertWavescale(spc,to='GHz')
  #
  # Correct for radial velocity of the Satellite (i.e. into the LSR frame)
  freq = spc.wave * (1.0-radialVelocity/c)
  spc.wave = freq
  #
  # Define the fitting model.
  # First, start with a N'th order polynomial (N=3) for the continuum
  N = 3
  polyInit = [0.0]*(N+1)
  sf = SpectrumFitter(spc,0,1,plotFit)
  sf.useFitter('lbm')
  M = sf.addModel('poly', [N], polyInit)
  models = [M]
  #
  # Then, add one sinc model per spectral line
  # Select only lines where there are data
  sel = lineFreqsCorr.where(lineFreqsCorr.gt(MIN(freq)).and(lineFreqsCorr.lt(MAX(freq))))
  #
  # Create a Sinc model for each line to be fitted
  for line in lineFreqsCorr[sel]:
   #print line
   d = MIN(ABS(freq - line))
   selLine = freq.where(ABS(freq - line) == d)
   ampInit = spc['flux'].data[selLine][0]
   initSinc = [ampInit, line, 1.1854189719256623/Math.PI]
   M = sf.addModel('sinc', initSinc)
   M.setFixed([2])
   models.append(M)
  # Carry out the global fit
  sf.doGlobalFit()
  # Calculate the residual
  sf.residual()
  residual = sf.getResidual()
  # Calculate the integrated line flux and print results
  for model in models[1:]:
   if lineCnt == len(lineFreqs): break
   parameters = model.getFittedParameters()
   parametersStdDev = model.getFittedStdDev()
   # calculate line characteristics only for the 12CO lines
   if lineFreqs[lineCnt] not in CO12Lines:
       lineCnt += 1
       continue
   else:
       lineIndex = CO12Lines.where(CO12Lines == lineFreqs[lineCnt]).toInt1d()[0]
   # Line flux in W/m2
   lineFlux = 1e-26*parameters[0]*1e9*Math.PI*parameters[2]
   LineFlux.set(j,i,lineIndex,obsCnt,lineFlux)
   #print hex(int(obsid)), parameters[1], lineFlux
   # Line flux error in W/m2
   # Original error calculation when sinc width is a free parameter
   #lineFluxError = lineFlux*SQRT((parametersStdDev[0]/parameters[0])**2+\
   #                          (parametersStdDev[2]/parameters[2])**2)
   # Error calculation when sinc width is NOT a free parameter:
   lineFluxError = lineFlux * parametersStdDev[0] / parameters[0]
   LineFluxError.set(j,i,lineIndex,obsCnt,lineFluxError)
   # Line sensitivity
   # Determine the RMS in the residual
```

```
        resWindow = freq.where( ((freq>(parameters[1]-resWinSize)) & (freq<(parameters[1]-5.0))) | \
                                ((freq<(parameters[1]+resWinSize)) & (freq>(parameters[1]+5.0))) )
        myRms = QRMS(residual[resWindow])
        SNR = parameters[0] / myRms
        LineSensitivity.set(j,i,lineIndex,obsCnt,lineFlux*(dt/1.0)*(5.0/SNR))
        # Line velocity in km/s
        lineVelocity = 1e-3*c*(lineFreqsCorr[lineCnt]-parameters[1])/lineFreqsCorr[lineCnt]
        LineVelocity.set(j,i,lineIndex,obsCnt,lineVelocity)
        # Line velocity error in km/s
        lineVelocityError = 1e-3*c*parametersStdDev[1]/lineFreqsCorr[lineCnt]
        LineVelocityError.set(j,i,lineIndex,obsCnt,lineVelocityError)
        lineCnt += 1
      pass
    pass
    obsCnt += 1
  pass # obsids
  i += 1
 pass # versions
 j += 1
pass # calibrations
obsCnt = 0
for obsid in obsids:
  SpireResultsLineSources['Line Flux %X'%obsid]=Column(data=LineFlux[0,0,:,obsCnt], \
      unit=(herschel.share.unit.Power.WATTS / herschel.share.unit.Length.METERS**2),description=src)
  SpireResultsLineSources['Line Velocity %X'%obsid]=Column(data=LineVelocity[0,0,:,obsCnt], \
      unit=(herschel.share.unit.Speed.KILOMETERS_PER_SECOND),description=src)
  if plotObs:
    # MAKE A PLOT FOR EACH OBSERVATION
    l  =  LayerXY(CO12Lines,LineFlux[0,0,:,obsCnt],  name="0x%X"%obsid,  line=0,  symbol=14, symbolSize = 5)
    pFlux.addLayer(l)
    pFlux.setXtitle('Line Frequency [GHz]')
    pFlux.setYtitle('')
    pFlux.getLegend().setVisible(1)
    l  =  LayerXY(CO12Lines,LineVelocity[0,0,:,obsCnt],  name="0x%X"%obsid,  line=0,  symbol=14, symbolSize = 5)
    pVel.addLayer(l)
    pVel.setXtitle('Line Frequency [GHz]')
    pVel.setYtitle('')
    pVel.getLegend().setVisible(1)
  obsCnt += 1
if printData:
    # PRINT AVG & STD DATA
    # INTEGRATED LINE FLUX
    avgLine = Double3d(len(calibrations), len(versions), len(CO12Lines))
    stdLine = Double3d(len(calibrations), len(versions), len(CO12Lines))
    avgVel = Double3d(len(calibrations), len(versions), len(CO12Lines))
    stdVel = Double3d(len(calibrations), len(versions), len(CO12Lines))
    #
    for k in range(len(calibrations)):
     for i in range(len(versions)):
      for j in range(len(CO12Lines)):
       avgLine.set(k,i,j,MEAN(LineFlux[k,i,j,:]))
       stdLine.set(k,i,j,STDDEV(LineFlux[k,i,j,:]))
       avgVel.set(k,i,j,MEAN(LineVelocity[k,i,j,:]))
       stdVel.set(k,i,j,STDDEV(LineVelocity[k,i,j,:]))
    #
    for oCnt in range(len(CO12Lines)):
        print '%4.1f   %4.1f   %4.1f   %2.1f  %2.1f'%(CO12Lines[oCnt],  1e+18*avgLine[0,0,oCnt], 1e+18*stdLine[0,0,oCnt], avgVel[0,0,oCnt], stdVel[0,0,oCnt])
         #100*stdLine[0,0,oCnt]/avgLine[0,0,oCnt])
if plotData:
    # MAKE COMPARISON PLOTS (AVG and STD)
```

```
      # INTEGRATED LINE FLUX
      avgLine = Double3d(len(calibrations), len(versions), len(CO12Lines))
      stdLine = Double3d(len(calibrations), len(versions), len(CO12Lines))
      #
      for k in range(len(calibrations)):
       for i in range(len(versions)):
        for j in range(len(CO12Lines)):
         avgLine.set(k,i,j,MEAN(LineFlux[k,i,j,:]))
         stdLine.set(k,i,j,STDDEV(LineFlux[k,i,j,:]))
      # Plot the average integrated line flux
      pSngFlux = PlotXY()
      for k in range(len(calibrations)):
       for i in range(len(versions)):
        l   =   LayerXY(CO12Lines,avgLine[k,i,:],   line=0,   symbol=14,   symbolSize=4,   name='%s
%s'%(versions[i],calibrations[k]))
        pSngFlux.addLayer(l)
        clr = l.getColor()
        pSngFlux.addLayer(LayerXY(CO12Lines,avgLine[k,i,:]-stdLine[k,i,:],    line=0,    symbol=20,
symbolSize=7, color=clr, inLegend=0))
        pSngFlux.addLayer(LayerXY(CO12Lines,avgLine[k,i,:]+stdLine[k,i,:],    line=0,    symbol=19,
symbolSize=7, color=clr, inLegend=0))
      pSngFlux.setXtitle('Line Frequency [GHz]')
      pSngFlux.setYtitle('')
      pSngFlux.setTitleText('Average Line Flux [W m-2] %s'%src)
      pSngFlux.getLegend().setVisible(1)
      #
      # LINE VELOCITY
      #
      avgLine = Double3d(len(calibrations), len(versions), len(CO12Lines))
      stdLine = Double3d(len(calibrations), len(versions), len(CO12Lines))
      #
      for k in range(len(calibrations)):
       for i in range(len(versions)):
        for j in range(len(CO12Lines)):
          avgLine.set(k,i,j,MEAN(LineVelocity[k,i,j,:]))
          stdLine.set(k,i,j,STDDEV(LineVelocity[k,i,j,:]))
      # Plot the average line velocities
      pSngFlux = PlotXY()
      for k in range(len(calibrations)):
       for i in range(len(versions)):
        l   =   LayerXY(CO12Lines,avgLine[k,i,:],   line=0,   symbol=14,   symbolSize=4,   name='%s
%s'%(versions[i],calibrations[k]))
        pSngFlux.addLayer(l)
        clr = l.getColor()
        pSngFlux.addLayer(LayerXY(CO12Lines,avgLine[k,i,:]-stdLine[k,i,:],    line=0,    symbol=20,
symbolSize=7, color=clr, inLegend=0))
        pSngFlux.addLayer(LayerXY(CO12Lines,avgLine[k,i,:]+stdLine[k,i,:],    line=0,    symbol=19,
symbolSize=7, color=clr, inLegend=0))
      pSngFlux.setXtitle('Line Frequency [GHz]')
      pSngFlux.setYtitle('')
      pSngFlux.setTitleText('Average Line Velocity [km s-1] %s'%src)
      pSngFlux.getLegend().setVisible(1)
      #
      # LINE SENSITIVITY
      #
      avgLine = Double3d(len(calibrations), len(versions), len(CO12Lines))
      stdLine = Double3d(len(calibrations), len(versions), len(CO12Lines))
      #
      for k in range(len(calibrations)):
       for i in range(len(versions)):
        for j in range(len(CO12Lines)):
          avgLine.set(k,i,j,MEAN(LineSensitivity[k,i,j,:]))
          stdLine.set(k,i,j,STDDEV(LineSensitivity[k,i,j,:]))
      # Plot the average line sensitivities
      pSngFlux = PlotXY()
      for k in range(len(calibrations)):
```

```
       for i in range(len(versions)):
        l   =   LayerXY(CO12Lines,avgLine[k,i,:],   line=0,   symbol=14,   symbolSize=4,   name='%s
%s'%(versions[i],calibrations[k]))
        pSngFlux.addLayer(l)
        clr = l.getColor()
        pSngFlux.addLayer(LayerXY(CO12Lines,avgLine[k,i,:]-stdLine[k,i,:],    line=0,    symbol=20,
symbolSize=7, color=clr, inLegend=0))
        pSngFlux.addLayer(LayerXY(CO12Lines,avgLine[k,i,:]+stdLine[k,i,:],    line=0,    symbol=19,
symbolSize=7, color=clr, inLegend=0))
      pSngFlux.setXtitle('Line Frequency [GHz]')
      pSngFlux.setYtitle('')
      pSngFlux.setTitleText('Average Line Flux Sensitivity [W m-2 hr 5sigma] %s'%src)

pSngFlux.addLayer(LayerXY(Double1d([14.9,25.5,32.0,35.5,46.7,51.5])*c/10000000,Double1d([2.94,0.9
4,2.04,1.56,1.56,2.15])*1E-17,line=1,stroke=2,name='OM',color=java.awt.Color.GRAY))
      pSngFlux.getLegend().setVisible(1)
  pass # calibrations
  #
  # Plot the sky positions
  if plotSkyPositions:
      pSky = PlotXY(titleText='%s (ODs %s)'%(src,days))
      lSkyNominal               =               LayerXY(Double1d([sds.meta['raNominal'].value]),
Double1d([sds.meta['decNominal'].value]),symbol=17,line=2,color=java.awt.Color.BLACK)
      pSky.addLayer(lSkyNominal)
      pSky.addLayer(LayerXY(SkyPosition_RA,
SkyPosition_DEC,symbol=14,line=0,color=java.awt.Color.BLUE))
      pSky.setPlotSize(3,3)
      pSky.setSize(800,800)
      pSky.setXtype(herschel.ia.gui.renderer.AxisConstants.Type.RIGHT_ASCENSION)
      pSky.setYtype(herschel.ia.gui.renderer.AxisConstants.Type.DECLINATION)
      pSky.setXtitle("RA [hh:mm:ss]")
      pSky.setYtitle("Dec [dd:mm:ss]")
pass # source

simpleFitsWriter(file='/spire/tmp/SpireResultsLineSources.fits', product=SpireResultsLineSources)
```