# DSN Radio Astronomy Single Dish FITS User's Guide

T. B. H. Kuiper

Revision of November 27, 2017

# Preface

**DSN SDFITS Convention**

**Command Line**

**Plotting**

**Programs**

**Monitor and Control**

**Appendix A** gives some suggestions for configuring a Linux workstation for reducing DSN radio astronomy spectral line data.

**Appendix B** gives a quick introduction to Python in which the M&C software is written. A convention used in this document is that if an object is shown capitalized (*e.g.* `Receiver` and in typewriter font, it is also a Python *class*. A class is a programming unit that serves as a template for objects that are similar to each other. In general, a single word in typewriter font refers to an attribute while one followed by empty parentheses is a method.

**Appendix C** descibes the circular polarization convention used in astronomy and how two orthogonal linear polarizations can be converted to counter-rotating circular polarizations.

**Appendix D** describes complex data samples can be generated and used to separate lower and upper sidebands.

# Contents

# List of Figures

# List of Code Examples

# Acronyms and Technical Terms

**ADC** - analog-to-digital converter.
**DAC** - digital-to-analog converter.
**IF** - intermediate frequency (signal)
**I/O** - input/output
**I/Q** - in-phase/quadrature-phase, the two components of a complex signal.
**K-band** - the frequency range 17-27 GHz.
**LO** - local oscillator.
**L/U** - lower sideband/upper sideband, a signal pair obtained from I/Q.
**MSB** - most significant bit (or byte, depending on context).
**RF** - radio frequency (signal)
**VLBI** - very long baseline interferometry
**WBDC** - wide-band down-converter, a class of receiver.
**WVSR** - wideband VLBI science receiver

# Chapter 1

# Introduction

The Deep Space Network conducts front-line radio astronomy research at its three deep space communication complexes (DSCC), near Canberra (Australia), Goldstone (California) and Madrid (Spain). Each Complex supports radio astronomers active in their host countries, as well as guest investigators from around the world. It is important for inter-operability that the DSN uses data file formats which are compatible with those used in those communities. At the same time, because of limited workforce resources, all Complexes must adhere to a common standard.

The Single Dish FITS convention [Liszt(1995)] (SDFITS) was developed around 1989, mostly for spectral line astronomy, and different implementations evolved at various institutions. A standard was registered in 2012 [Pence(2012)]. However, interpretations are not uniform resulting in SDFITS files that have different structures, such as those from the Green Bank Telescope and the Parkes and MOPRA telescopes.

The goal of the DSN version of SDFITS is to allow full recording of spectral line data and metadata from a wide variety of backends. The general philosophy is that anything which does not violate the accepted SINGLE DISH standard is allowed. This means that to reduce data with an existing package such as ASAP, CLASS, or GBTIDL will probably require a (probably quite simple) conversion from a DSN SDFITS file to one adhering to a different convention.

This document describes tools available for using Python to reduce and analyze data in DSN SDFITS format.

# Chapter 2

# DSN SDFITS Convention

## 2.1 Overview

This summarizes the features of the DSN convention for SDFITS.

**Multidimensional columns**

**The `CYCLE` column**

**One extension per spectrometer**

**A `PCG TONES` extension for calibration tones**

**Uncalibrated `TSYS` proxy**

Additional features under consideration are an extension for extensions for ambient-load/noise-diode calibrations and for tipping curves and non-degenerate coordinate axes (axes 2 and 3).

## 2.2 Multidimensional Columns

The main difference between the DSN and other conventions is that any table column may be multi-dimensional as long as the meaning and order of the axes, except the first, is the same. To illustrate this, consider a receiver with two feeds, each having two polarized outputs and a backend which produces ten 4096 channel spectra per scan. The DSN convention follows the NRAO one in that the DATA column would have `TDIM(4096,1,1,2,10,2)`. If a system temperature is measured for each IF and spectrum, the `TSYS` column would have cells which are numpy arrays with dimension $(1 \times 1 \times 1 \times 2 \times 10 \times 2)$. Note that this column does not require a separate `TDIM` because all the axis meanings are the same and the length of the first axis can be obtained from the `array.shape` attribute.

# Chapter 3

# Command Line

## 3.1 Overview

While one may generally want to use a Python program to avoid repetitive actions, for simple tasks the Python command line is often the quickest and clearest way to obtain results. It also serves as a convenient way to recall attributes and methods of classes used in data reduction. And, it is a convenient way to introduce the classes.

## 3.2 Class `SessionAnalyzer`

Snippet 3.1 introduces the main class for analyzing data from an observing session. When invoked without arguments the class initialization will query for the observing session details. The project ID is only used for knowing where to store any data reduction results. It is not used in the math to the FITS file.

Initiating a `SessionAnalyzer` causes each of the FITS files for the session to be assigned a `DSNFITSplotter` object which opens the file for data reduction.

## 3.3 Class `DSNFITSplotter` and `DSNFITSexaminer`

`DSNFITSplotter` is actually a subclass of (*i.e.* is derived from) `DSNFITSexaminer`, to which it adds plotting capability. The real work of data reduction is done by `DSNFITSexaminer`.

### 3.3.1 Key Attributes

The main attributes of the `DSNFITSexaminer` class are:

**`file`** which is the path to and name of the FITS file,

```
kuiper@kuiper:~$ ipython
...
In [1]: from Data_Reduction.SLATool import SessionAnalyzer
...
In [3]: sa = SessionAnalyzer()
0 > dss14
1 > dss43
Select a station by index: 1
0 > 2001
...
6 > 2015
7 > 2017
Select a year by index: 6
0 > 204
...
7 > 242
Select a day BY INDEX: 0
In [4]: sa?
Type:       SessionAnalyzer
String Form:<Data_Reduction.SLATool.SessionAnalyzer object at 0x7f630d89fc90>
File:       /usr/local/lib/python2.7/DSN-Sci-packages/Data_Reduction/SLATool.py
Docstring:
Tool for reducing multiple data reduction sessions

Example::
  In [1]: from Data_Reduction.SLATool import SessionAnalyzer
  In [2]: sa = SessionAnalyzer(project='67P', year=2015, DOY=204)
  In [3]: x, sum_y, sum_Tsys, sum_intgr = sa.get_average()
Constructor Docstring:
initiate a SessionAnalyzer

@param project : name as in /usr/local/projects directory
@type  project : str
@param dss : DSN station
@type  dss : int
@param year : of observing session
@type  year : int
@param DOY : of observing session
@type  DOY : int
```

**Snippet 3.1:** The SessionAnalyzer class.

**`hdulist`** of the header and data units in the file which can examined with `ex0.hdulist.info()`,
**`header`** for the FITS file,

`tables` which is a Python `dict` of the SINGLE DISH binary tables in the file.

Usually, a FITS file has only one extension with a table but the DSN convention allows multiple backends and each backend must have its own extension.

### 3.3.2 Tables

Tables are implemented as classes private to the `DSNFITSexaminer` class, that is, they are defined within the class. The real work of data reduction is done with tables. This class is still under active development (as of November 27, 2017) but some of the main features should be stable by now. Snippet 3.2 shows the first thing one might want to do when a table has been opened. The convention used with the SAO[1]/TAMS[2] is to have a separate FITS file for each

```
In [6]: tb0.make_directory()
Row Scan ch      Source       Sig  Freq      intg
--- ---- -- ---------------- ----- --------- ----
  0    1  1         Orion-KL  True  22000 U   65
  1    2  1         Orion-KL False  22000 U   65
```

**Snippet 3.2:** Directory of a FITS file binary table.

source observed as a convenience for data reduction. On the other hand hand, all the data recorded from all the sources in a spectroscopy session using the WVSR backend goes into one FITS file. Then, a method for selecting rows from a table is useful. Snippet 3.3 illustrates this. This illustrates the use of the `CYCLE` keyword adopted from the ATNF convention for SDFITS.

#### Operating on Table Rows

To fetch the spectra from a selection of rows, there is the `Table` method `get_spectra()`. How the spectra are handled depends on the observing mode.

The dataset shown in Snippet 3.3 were taken in the `LINEPSSW` mode. The conventional way to reduce these data is to form normalized difference spectra ($S_N$) from the on-source ($S_{sig}$) and off-source ($S_{ref}$) spectra with

$$S_N = \frac{S_{sig} - S_{ref}}{S_{ref}}.$$

Snippet 3.4 implements this. The resulting data cube has two rows, one for each pair of the original ON and OFF rows, with four Stokes parameter spectra of 8192 points.

[1]Smithsonian Astrophysical Obervatory
[2]Tidbinbilla AGN Maser Survey

```
In [2]: tb = sa.examiners[0].tables[0]
In [3]: tb.make_directory()
Row Scan ch      Source       Sig  Freq      intg
--- ---- -- ---------------- ----- --------- ----
  0    1  1           Mon_R2  True   8309 U   90
  1    1  2           Mon_R2  True   8585 U   90
  2    2  1           Mon_R2 False   8309 U   90
  3    2  2           Mon_R2 False   8585 U   90
  4    3  1           Mon_R2  True   8309 U   90
  5    3  2           Mon_R2  True   8585 U   90
  6    4  1           Mon_R2 False   8309 U   91
  7    4  2           Mon_R2 False   8585 U   91
...
220  111  1       ros-fregg3  True   8309 U   90
221  111  2       ros-fregg3  True   8585 U   90
222  112  1       ros-fregg3 False   8309 U   12
223  112  2       ros-fregg3 False   8585 U   12
224  124  2       ros-fregg3 False   8585 U    1
225  128  2                  False      0 None    0
In [5]: tb.select({'OBJECT': 'Mon_R2', 'CYCLE': 1})
Out[5]: [0, 2, 4, 6]
In [6]: tb.select({'OBJECT': 'Mon_R2', 'CYCLE': 2})
Out[6]: [1, 3, 5, 7]
```

**Snippet 3.3:** irectory of a FITS file binary table obtained with a WVSR configured for two frequency channels.

```
In [1]: run interactive.py --dss=14 --date=2017/010
In [2]: tb = sa.examiners[0].tables[0]
In [3]: rowsON = tb.select({'OBJECT': 'Mon_R2', 'CYCLE': 1, 'SIG': True})
In [4]: rowsOFF = tb.select({'OBJECT': 'Mon_R2', 'CYCLE': 1, 'SIG': False})
In [5]: normalized = (tb.get_spectra(rowsON)-tb.get_spectra(rowsOFF)) \
                     /tb.get_spectra(rowsOFF)
In [8]: normalized.shape
Out[8]: (2, 4, 8192)
```

**Snippet 3.4:** Reducing position-switched spectral line data.

The data shown in Snippet 3.2 were taken in the `LINEPBSW` mode. In the `SIG=True` part of the switch cycle (a pair of scans) the source is in beam 1; in the `SIG=FALSE` part, the source is in beam 2. This is an attractive mode because there is always a beam taking on-source data. As Snippet 3.5 shows, the reduction is the same. In this case there is only one row (obtained from the original rows 0 and 1) and the data cube has has two values on the `BEAM` axis, seven

```
In [8]: rowsON = tb0.select({'OBJECT': 'Orion-KL', 'SIG': True})
In [9]: rowsOFF = tb0.select({'OBJECT': 'Orion-KL', 'SIG': False})
In [10]: normalized = (tb0.get_spectra(rowsON)-tb0.get_spectra(rowsOFF)) \
                      /tb0.get_spectra(rowsOFF)
In [11]: normalized.shape
Out[11]: (1, 2, 7, 2, 32768)
```

**Snippet 3.5:** Position switched reduction of beam-switched data.

records on the `TIME` axis, and two polarizations with a 32768 channel spectrum for each. For the first beam the spectrum is what one would expect but for the second beam it is negative. So, in a final step, one computes (beam$_1$-beam$_2$/2).

### `Table` Methods for Data Reduction

There are many `Table` methods to support data reduction. We highlight a few here:

**`get_table_stats()`** gets the number of scans, cycles, rows and observing modes used in the table;

**`get_indices(scan=1, cycle=1, pol=1, beam=1, record=1, trimmed=False)`** returns indices for getting one spectrum from SPECTRUM column;

**`get_index_keys()`** which returns labels that can be associated with the index keys, such as `row`, `beam`, `record`, `pol`, `dec`, and `RA`;

**`freqs(row=0)`** computes frequencies for spectra in the specified row in MHz based on the values of `CRVAL1`, `CDELT1`, and `CRPIX1`;

**`normalized_beam_diff(rows)`** computes (on-source-off-source)/off-source for each record and each pol for spectrometers with data from two or more beams;

**`BPSW_spectra(rows, Tsys=None, weighted=False)`** returns normalized record-by-record on-beam minus off-beam for data with a `TIME` azis;

**`BPSW_average(rows, weighted=False, TAMS_hack=True)`** produces scaled, averaged spectra for each scan pair[3]; and,

**`scans_average(rows, weighted=True, TAMS_hack=True)`** averages the scans weighted by int. time and $1/T_{sys}^2$.

With the TAMS spectrometer, the entire 32,768 channel 1020 MHz spectra are seldom useful and so there is a method to extract data subsets into `Data` class objects.

**Class `Data`** is a private class of the `Table` class consisting of a set X-Y data with associated attributes and methods for subsets of a full spectrum.

[3] `TAMS_hack` uses beam 2 for both the SIG and REF $T_{sys}$ because the beam 1 data are not available.

**`extract_window(rows, data=None, Tsys=None, intgr=None, xlimits=(0,32767)`** extracts a subset of a SPECTRUM column. Additional arguments are `frame="CHAN-OBS"` and `source="67P")`. The datacube can be specified by a list of rows with the given X-axis limits, in which case the spectra will be extracted from the SPECTRUM column. Or, a data cube returned from `normalized_beam_diff()`, ' `BPSW_spectra()`, `BPSW_average()`, or `scans_average()` can be provided. In the case of latter two averaging methods, the averaged system temperatures and integration times returned by these methods can also be provided. The X limits are in units appropriate to the frame. The corresponding channel numbers will be computed as `Data` attributes.

**`rel_freq_units(frame="FREQ-OBS", ref_freq=None, v_frame=0, row=0)`** used by `extract_window()` to compute the X-axis units for various reference frames.

### Computing Atmospheric Opacity

The `Table` class also has methods for analyzing the implicit tipping curve data provided by the system temperatures and telescope elevations recorded while taking spectral line data.

**`validate_wx_data()`** ensure that the tipping curve data are usable by making sure that at least some are good and creates a mask for those.

**`get_wx_datacubes()`** creates data for analyzing environmental conditions. It returns a dict with keys `UNIXtime`, `ELEVATIO`, `TSYS`, `TAMBIENT`, `PRESSURE`, `HUMIDITY`, `WINDSPEE` and `WINDDIRE`. The data asociated with each key is a dict with numpy array for `SIG` with values `True` and `False`. The `TSYS` array has four axes representing time index(a 0-based sequence in order of matplotlib), datenum (`matplotlib.dates` time), subchannel (CYCLE value), beam (1-based number sequence) and IF (1-based number sequence, usually representing the POL axis value). The other keys have only a time axis.

**`fit_mean_power_to_airmass(Tvac_func, first=0, last=None, replace=False)`** fits the mean power data *vs* airmass to the radiative transfer equation. This assumes that every IF has a way of measuring power. The measured power is a single value along the first axis of the data array (or last index in a C/Python array). If there are multiple records, *i.e.* a `TIME` axis then they will be averaged.

It turns out that for the DSN spectrometers, the power averaged over the entire spectrum is a good proxy for system temperature, barring a scaling factor. So when power-meter-based system temperatures are not available this average spectrum power is saved in the `TSYS` column but with the column unit set to `count` instead of K.

The scaling factor can be estimated if one has a previously determined zero airmass value for system temperature. If one is confident about that procedure then setting the `replace` argument to `True`.

# Chapter 4

# Plotting

# Chapter 5

# Programs

# Chapter 6

# Monitor and Control System

## 6.1 Overview

The DSN Radio Astronomy monitor and control system is designed for conducting observations. However, sometimes knowlege of the classes defined within the M&C system and their attributes (*i.e.* variables) and methods (*i.e. functions*) is helpful for analyzing data. This chapter touches on the relevant features.

# Appendix A

# Workstation Configuration

To be done as `root`. The secure way to do that is to be in the `sudoers` file with ALL privilege. Then do `sudo su -`. The reason for doing this instead of prefixing all the commands with `sudo` is that paths and permissions are different and some commands may croak at installing files in some places.

```
apt-get install mlocate
apt-get install git
apt-get install apache2 php5 libapache2-mod-php5
apt-get install pyro
apt-get install python-setuptools
apt-get install python-pip
apt-get install python-epydoc
pip install dill
pip install pyephem
pip install openpyxl==1.5.8
```

The purposes of all these are:

**mlocate** to build and maintain a searchable database of files.
**git** for software management.
**apache2** for web-based interfaces to the equipment servers.
**pyro** for inter-process communication in Python.
**setuptools** to facilitate packaging and installing Python projects.
**pip** to install and maintain packages in the Python Package Index.
**epydoc** for documenting Python source code.
**dill** , a superior version of `pickle` with less restrictions.
**pyephem** to provide a superclass for `DSS` in `Configurations`.
**openpyxl** to use Excel spreadsheets for configuration information and data.

When these packages are installed one can bring in the Python monitor and control software. All locally developed packages are in `/usr/local/lib/python2.7/DSN-Sci-packages/`. To put this directory in the Python path the file `DSN-Sci-packages.pth` is put into `/usr/local/lib/python2.7/dist-packages`. It contains one line:

```
kuiper@kuiper:/usr/local/lib/python2.7/dist-packages$ cat DSN-Sci-packages.pth
/usr/local/lib/python2.7/DSN-Sci-packages
```

The primary Git remote is at `https://ra.jpl.nasa.gov/`. This is a password-protected remote which can be used by our overseas partners who have access to the JPL domain but not `github.jpl.nasa.gov` because they are not "US persons".

Because Gitlab does not support `gh-pages`, a method for publishing web pages for a package, we also use, for now, `github.jpl.nasa.gov` to provide these pages.

In the examples below, note the directories in which the repos are cloned. This applies to all DSN Radio Astronomy hosts. The directory `/usr/local` must be recursively writable by group `JPLusers`. For setting up ssh tunnels and to provide a customized version of `epidoc`.

```
/usr/local/$ git clone https://ra.jpl.nasa.gov/RadioAstronomy/scripts.git
```

Package `support` provides a variety of general purpose modules.

```
/usr/local/lib/python2.7/DSN-Sci-packages$ git clone \
                           https://ra.jpl.nasa.gov/RadioAstronomy/support.git
```

For date and time conversions, probably needed for writing data files,

```
/usr/local/lib/python2.7/DSN-Sci-packages$ git clone \
                         https://ra.jpl.nasa.gov/RadioAstronomy/DatesTimes.git
```

The core of the monitor and control package is `MonitorControl`:

```
/usr/local/lib/python2.7/DSN-Sci-packages$ git clone \
                     https://ra.jpl.nasa.gov/RadioAstronomy/MonitorControl.git
```

Most servers manage some kind of equipment for which there are packages in

```
/usr/local/lib/python2.7/DSN-Sci-packages$ git clone \
                        https://ra.jpl.nasa.gov/RadioAstronomy/Electronics.git
```

As described in `https://github.jpl.nasa.gov/pages/RadioAstronomy/Overview/`, modules to support specific hardware are in sub-directories. Only those modules required for the hardware attached to the server need to be installed. For example, for Radipower power meters:

```
/usr/local/lib/python2.7/DSN-Sci-packages/Electronics/Instruments$ git clone \
       https://ra.jpl.nasa.gov/kuiper/Electronics_Instruments_Radipower.git Radipower
```

Note the difference with respect to the earlier clones:

- The `clone` is done in a sub-directory of the top-level repo `Electronics`.
- It is cloned into a sub-directory named for the last part of the repo name.

The repo names reflects the directory structure `Electronics/Instruments/Radipower` as well as the Python module import path `Electronics.Instruments.Radipower`

The configuration information for the DSN Complex where the server is to be installed must also be cloned. For example

```
/usr/local/lib/python2.7/DSN-Sci-packages/MonitorControl/Configurations$ git clone \
  https://ra.jpl.nasa.gov/RadioAstronomy/MonitorControl_Configurations_CDSCC.git CDSCC
```

# Appendix B

# Un Soupçon de Python

The DSN Radio Astronomy Monitor and Control system uses the object-oriented features of Python. Arguably, object-oriented program can be said to be based on Plato's theory of forms. Python objects are analogous to real objects while classes correspond to forms or ideals, which are abstractions based on real objects.

## B.1 `iPython`

One can simply type `python` at a shell prompt to get the standard Python command line interface. However, `ipython` is preferable because it has powerful extra features. One that is used most often is *tab completion* illustrated in Snippet B.1. This is very useful if you can't

```
In [12]: sa.<Tab>
sa.DOY                    sa.get_good_weather_data  sa.project
sa.DSS                    sa.get_sources            sa.projectdatapath
sa.consolidate            sa.logger                 sa.projworkpath
sa.datapath               sa.open_datafiles         sa.sources
sa.examiner_keys          sa.plot_elev_and_Tsys     sa.year
sa.examiners              sa.plot_weather
sa.get_average            sa.plot_wind
```

**Snippet B.1:** Use of `<Tab>` to see the names of an object's attributes and methods.

remember the name of an attribute or method.

## B.2 Classes

A class is a natural way of describing a real-world object which has properties (*attributes*) and can do things (*methods* or functions). The most basic class in the M&C system is `Device` which has the attributes `name`, `inputs` and `outputs`, which are instance of `Port` classes, and `data` about the device, such as the location of a telescope or the bandwidth of a receiver. Most classes discused here are sub-classes of `Device`, which means that they "inherit" these attributes.

In the hope that it is more enlightening than confusing, Snippet B.2 (page 22) gives an example of a class definition. The `__init__()` method creates an instance of this class[1], which

```
class Telescope(Device):
  def __init__(self, obs, dss=0, LO=None, active=True):
    name = "DSS-"+str(dss)
    Device.__init__(self, name)
    self.inputs = {obs.name:obs}
    self['longitude'], self['latitude'], self['elevation'], tz, name, diam = \
            get_geodetic_coords(dss=int(dss))
    self['geo-x'],self['geo-y'],self['geo-z'] = \
            get_cartesian_coordinates('DSS '+str(dss))
    self.outputs[self.name] = Port(self, self.name, signal=Beam(str(dss)))
    self.outputs[self.name].signal['dss'] = dss
```

**Snippet B.2:** Stripped-down definition of the `Telescope` class.

inherits attributes from the class `Device`. The second line creates the object and the rest of the code assigns values to its attributes.

[1] Creating an object from a class is called *instantiation*, making an `instance` of a class.

# Appendix C

# Polarization

The DSN receivers are the older three-channel radio astronomy K-band receivers have native circular polarization. The default polarization of the new four-channel K-band receiver is linear. In order to average gain variations in the receiver channels over both measured polarizations, it is sometimes desirable to convert from one polarization mode to the other. This appendix gives the official definition of circular polarization and the technique for changing polarization mode.

## C.1 IAU Definition of Circular Polarization

As reported in the Proceedings of the Fifteenth General Assembly [IAU(1974)], the following resolution was adopted by IAU Commissions 25 (Stellar Photometry and Polarimetry) and 40 (Radio Astronomy):

*RESOLVED, that the frame of reference for the Stokes parameters is that of Right Ascension and Declination with the position angle of the electric-vector maximum, $\theta$, starting from North and increasing through East. Elliptical polarization is defined in conformity with the definitions of the Institute of Electrical and Electronics Engineers (IEEE Standard 211)*[IEEE(1969)].

**Left-Handed (Counterclockwise) Polarized Wave** (LCP): An elliptically polarized electromagnetic wave in which the rotation of the electric field vector with time is counterclockwise for a stationary observer looking in the direction of the wave normal.

**Right-Handed (Clockwise) Polarized Wave** (RCP): An elliptically polarized electromagnetic wave in which the rotation of the electric field vector with time is clockwise for a stationary observer looking in the direction of the wave normal.

This means that the polarization of incoming radiation for which the position angle of the electric vector measured at a fixed point in space increases with time is described as right-handed and positive. This can be seen in the lower panel of Figure C.1[1]

[1] To view these images relax the eye muscles as if looking at a distant object. A person not experienced in this

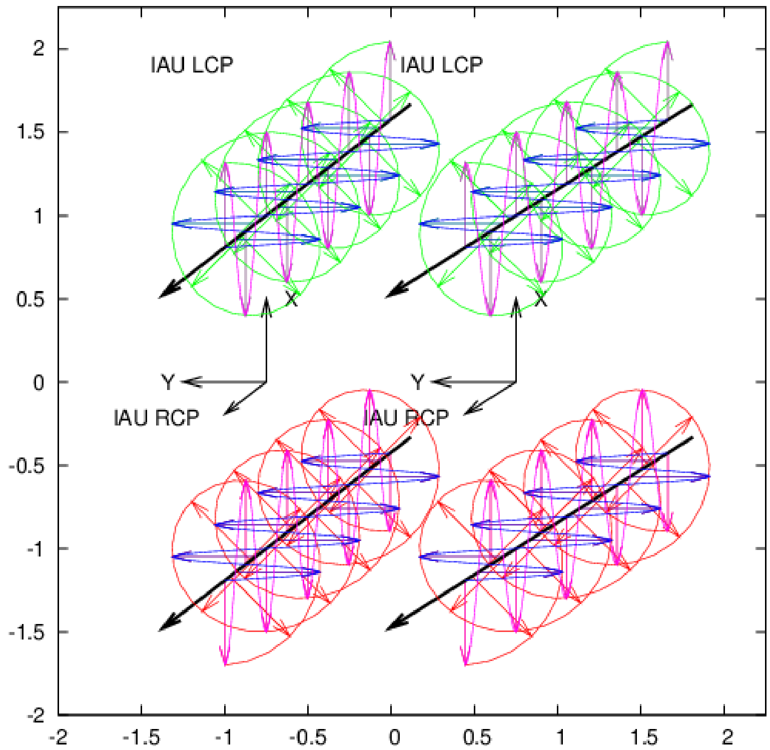


Figure C.1: A right circularly polarized wave (LCP - green) and left circularly polarized wave (RCP - red) as defined by the IAU are shown decomposed into two orthogonal linearly polarized waves (X - magenta, Y - blue). The thick arrows show the direction of propagation.

## C.2 Changing Between Linear and Circular Polarization

Figure C.1 shows that linear polarization can be converted to circular polarization. If the vertical (H-plane or Y-axis) wave lags behind the horizontal (E-plane or X-axis) wave by 90° of phase delay the resulting polarization will be RCP. If the vertical (H-plane or Y-axis) wave leads the horizontal (E-plane or X-axis) wave by 90° of phase delay the resulting polarization will be LCP.

Optically, this can be accomplished by passing the beam through a particular thickness of material which has different indices of refraction for two orthogonally polarized components.

type of viewing might first look at a distant object and then slide the gaze down to the figure without refocusing. Another method is to place a sheet of cardboard with one edge down the middle of the figure and the other edge between the eyes. It is probably easier to focus first on a label above or below the spiral, and then slide the gaze downwards.

Similarly, a waveguide section can be produced to perform the same function.

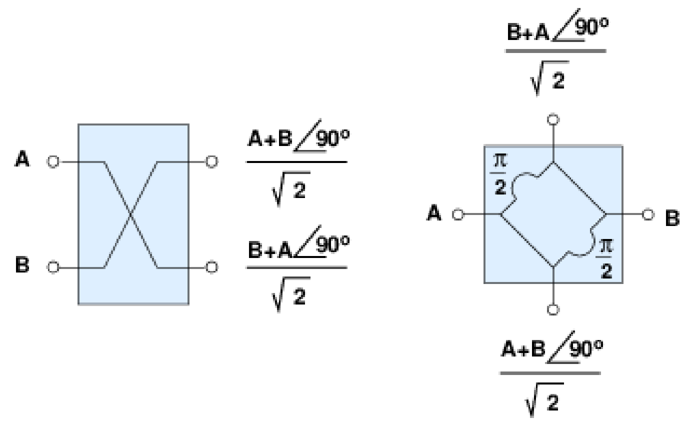


Figure C.2: A quadrature hybrid combines two input signals so that one is delayed 90° with respect to the other. There are two outputs which are symmetrically delayed. The figure on the left shows a conventional schematic. The one on the right is a more illustrative schematic.

In the cm, mm, and submm wave domains, there exist devices which can separate a radio wave into two orthogonally polarized components. A quadrature hybrid can convert between the two modes, as shown in Figure C.2. If port A were receiving vertical polarization (H-plane or Y-axis) and B horizontal polarization (E-plane or X-axis) then at the port labelled A+$\angle$90°B, the vertical polarization lags behind the horizontal and it would put out RCP. The port labelled B+$\angle$90°A would put out LCP.

From this one can see that, if there is an option to bypass the hybrid, it depends which way the hybrid is put in the circuit which determines whether the output ports are E/L and H/R or are E/R and H/L.

# Appendix D

# Sideband Separation

If the LO is in the middle of the RF band, then the mixing products will have both positive frequencies from the upper sideband and negative frequencies from the lower sideband. Both are present in the IF. A complex mixer puts out two IFs which are separated 90° in phase. These two IFs can be considered the real and imaginary parts of a complex signal. The upper and lower sidebands can be recovered from a complex signal.

The outputs from a complex mixer can be phased together to separate the upper and lower sidebands. This is done with a 90° hybrid, shown in Figure C.2, which re-combines the two quadrature phase mixing products. To see this, consider the outputs of the two mixers to be the sum of the USB and the LSB part of the down-converted spectrum with the frequencies explicitly positive and negative, *i.e.*,

$$\begin{aligned} V_R &= \frac{V_U}{\sqrt{2}}\cos(\omega_U t + \Delta\phi_U) + \frac{V_L}{\sqrt{2}}\cos(-\omega_L t + \Delta\phi_L), && \text{(D.1)} \\ &= \frac{V_U}{\sqrt{2}}\cos(\omega_U t + \Delta\phi_U) + \frac{V_L}{\sqrt{2}}\cos(\omega_L t - \Delta\phi_L), && \text{(D.2)} \\ V_I &= \frac{V_U}{\sqrt{2}}\sin(\omega_U t + \Delta\phi_U) + \frac{V_L}{\sqrt{2}}\sin(-\omega_L t + \Delta\phi_L), && \text{(D.3)} \\ &= \frac{V_U}{\sqrt{2}}\sin(\omega_U t + \Delta\phi_U) - \frac{V_L}{\sqrt{2}}\sin(\omega_L t - \Delta\phi_L) && \text{(D.4)} \end{aligned}$$

where $U$ refers to the upper sideband and $L$ the lower sideband. Then if these signals are combined in a quadrature hybrid, the effect along the paths with the $\pi/2$ delay is that the delayed signal at the summation point is that which entered the delay section $t - \pi/2\omega$ earlier. This means that we subtract $\pi/2$ from the angular terms in Equations D.1 and D.3 as illustrated in Figure D.1.

Subtracting $\pi/2$ from the angular terms is equivalent to subtracting $\pi/2$ from the phases associated with the positive frequency terms and adding $\pi/2$ to the phases associated with the

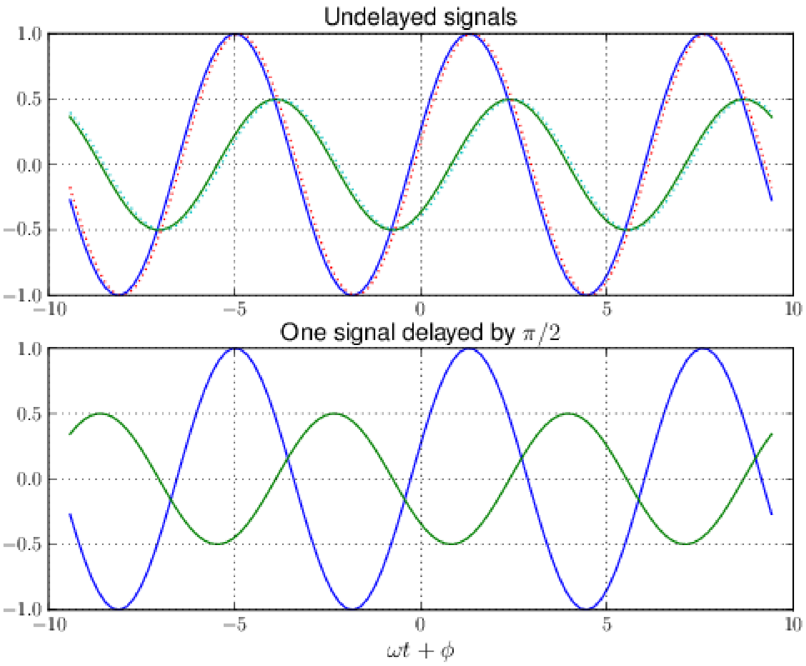


Figure D.1: The angular term for the green line in the lower panel is $\pi/2$ smaller than in the upper panel, showing that the effect is to delay the signal. (The dotted lines show a slighly earlier time for a positive frequency signal; so the signal moves left with time.)

negative frequency terms in Equations D.2 and D.4. The outputs from the hybrid will then be

$$
\begin{aligned}
\frac{V_R}{\sqrt{2}} + \frac{V_I\angle 90^\circ}{\sqrt{2}} &= \frac{V_U}{\sqrt{2}}\cos(\omega_U t + \phi_U) + \frac{V_L}{\sqrt{2}}\cos(\omega_L t - \phi_L) \\
&\quad + \frac{V_U}{\sqrt{2}}\sin(\omega_U t + \phi_U - \frac{\pi}{2}) - \frac{V_L}{\sqrt{2}}\sin(\omega_L t - \phi_L - \frac{\pi}{2}) \\
&= \frac{V_U}{\sqrt{2}}\cos(\omega_U t + \phi_U) + \frac{V_L}{\sqrt{2}}\cos(\omega_L t - \phi_L) \\
&\quad - \frac{V_U}{\sqrt{2}}\cos(\omega_U t + \phi_U) + \frac{V_L}{\sqrt{2}}\cos(\omega_L t - \phi_L) \\
&= \sqrt{2}V_L\cos(\omega_L t + \phi_L) \qquad \text{(D.5)} \\
\frac{V_R\angle 90^\circ}{\sqrt{2}} + \frac{V_I}{\sqrt{2}} &= \frac{V_U}{\sqrt{2}}\cos(\omega_U t + \phi_U - \frac{\pi}{2}) + \frac{V_L}{\sqrt{2}}\cos(\omega_L t - \phi_L - \frac{\pi}{2})
\end{aligned}
$$

$$\begin{aligned}
& +\frac{V_U}{\sqrt{2}}\sin(\omega_U t+\phi_U)-\frac{V_L}{\sqrt{2}}\sin(\omega_L t-\phi_L) \\
= & \frac{V_U}{\sqrt{2}}\sin(\omega_U t+\phi_U)+\frac{V_L}{\sqrt{2}}\sin(\omega_U t-\phi_L) \\
& +\frac{V_U}{\sqrt{2}}\sin(\omega_U t+\phi_U)-\frac{V_L}{\sqrt{2}}\sin(\omega_U t-\phi_L) \\
= & \sqrt{2}V_U\sin(\omega_U t+\phi_U) \qquad \text{(D.6)}
\end{aligned}$$

Note that adding $\pi/2$ to the image band terms in Equations D.1 and D.3 is like subtracting $\pi/2$ from the image band terms in Equations D.2 and D.4, *i.e.* the sign of the $\pi/2$ added is the same as the sign of the frequency.

The time-averaged power in each band is

$$\begin{aligned}
P_{USB} &= V_{USB}^2 \\
&= 2V_U^2\left\langle\cos^2(\omega_{IF}t+\phi_U)\right\rangle \\
&= V_U^2 \qquad \text{(D.7)} \\
P_{LSB} &= V_L^2 \qquad \text{(D.8)}
\end{aligned}$$

Equations D.7 and D.8, when added together, give the power in both sidebands.

# Bibliography


[IAU(1974)] IAU. 1974, in Proc. IAU General Assembly 15, IAU (Dordrecht: Reidel), 166

[IEEE(1969)] IEEE. 1969, IEEE Transactions on Antennas and Propagation, AP-17, 270

[Liszt(1995)] Liszt, H. S. 1995, NRAO preprint

[Pence(2012)] Pence, W. D. 2012, Single Dish FITS (SDFITS) Convention for Radio Astronomy Data, http://fits.gsfc.nasa.gov/registry/sdfits.html