# Reducing spectral line data

The raw data (in MBFITS format) are converted to CLASS format automatically. This is done using the method described in Winkel, Kraus, & Bach, *Unbiased flux calibration methods for spectral-line radio observations*, A&A 540, A140, 2012 (Section 3.4 and Section 4.5) - assuming a *constant system temperature* (Tsys) and *flat calibration signal* (Tcal). The latter is produced by a noise diode the power of which is in most cases directly fed into the waveguide within the receiver. *Neglecting the frequency-dependence of Tsys and Tcal introduces a bias, especially for wide-band observations!* (For more information see Winkel, Kraus, & Bach 2012 where we also show ways to do it better.)

The data files can be found on the *observer4* or *observer3* in the directory **/daten/Class/class_YYYY_MM_DD.100m**. There might be a short delay until the data is processed and is visible in the class file after the observation has finished

**Spectra in Class are always in units of Tcal!** To obtain Kelvins or Jy/Beam you have to multiply with the temperature of the noise diode (Tcal) and appropriate antenna sensitivities/efficiencies. Approximate values for this can be found on our receiver website receiver website. However, it is highly recommended to do calibration measurements during your observation for better accuracy. Furthermore, the automatic Class pipeline does not account for atmospheric opacity, elevation-gain curve, and antenna efficiency. Information about these steps can be found in Kraus 2009 (calibration memo).

Of course it is also possible to use the raw data directly, should one wish to do so.

# Using class

To reduce your (spectroscopic and pointing) data with `class` one should use the or **observer2** or **observer4** computer. The latter is in the MPIfR LDAP environment, i.e., everyone with an account in the MPIfR network can login to this computer with his/her account. It is connected to the `/homes` server. It is also connected to `/homes/astro/gag`, so your gildas package of choice should work. It is also possible to use the Effelsberg version of class that is used to write the spectroscopic and pointing data. To enable this, change to the bash shell by typing bash and source the init file:

```
source /opt/specpipeline/init_classwriter.sh
```

afterward you can run

```
class
```

The Data from the Class pipeline is stored in `/daten/Class`. To open the file with Class use

```
las90> file in "/daten/Class/class_2010_10_20.100m"
las90> find
```

For pointings you have to switch to continuum mode first:

```
las90> set type c
```

```
las90> find
```

One can switch back to line observations with:

```
las90> set type l
las90> find
```

To look for new data in the file:

```
las90> new
```

(Note: This doesn't always work. In such cases simply reopen the file, using `file open`.)

Printing is possible using:

```
las90> hardcopy /dev eps color /plot HP-RECH
```

In some cases, a file permission error may occur, in that case create a temporary eps file, please

```
las90> hardcopy "/tmp/tmp.eps" /dev eps color /plot HP-RECH /overwrite
```

A list of available printer names is obtained using

```
sh> lpstat -p -d
```

For more information on how to use Class have a look into the Gildas documentation.

# Frequency switching

Note that in frequency switching mode the Class pipeline already performs the folding (since the `fold` command of Class produces incorrect results for some cases). Details can be found in the News section and Winkel, Kraus & Bach 2012.

# Reprocessing spectra using the Class pipeline

Sometimes it may be necessary to re-reduce the data with the Class pipeline. This can be done on the observer4 computer. The necessary software is installed in `/opt/specpipeline`. There is a bash script to set all the necessary paths. "tcsh" shell users should change to "bash" by typing `bash`. Launch the following command:

```
source /opt/specpipeline/init_classwriter.sh
```

Now you should be ready to run the offline Class calibration pipeline. There are two ways to go, an interactive tool and a command-line tool (for simple tasks).

## Interactive mode - OfflinePipeline

Type

```
OfflinePipeline
```

Now you are in an ipython interface to reduce the data. There are the following commands to reduce the data:

```
setFitsDir('/daten/Raw')
```

which sets the source directory for the rawdata. The program will look in this directory for the raw mbfits files.

```
setClassName('Filename','Directory')
```

This sets the output gildas class file. If only:

```
setClassName('Filename')
```

is given, the named file is created in the current dictionary. Without this command a file name e.g. like `class_2010_10_20.100m` is created in the current directory. To write the files one has to know the scan number and must give the subscan number. Subscan numbers are 1 based. If the subscan is not in the file an error is plotted.

```
reduceSubscan(scannumber, subscannumber)
```

If you reduce the same scan twice, it is written twice to the class file with the same scan number (which you usually don't want).

### Batch processing

For projects involving lots of scan numbers, one may want to speed up things by using a small piece of python code such that one doesn't have to type in all scan numbers manually:

```
setClassName('myfile.100m')
setFitsDir('/daten/Raw')

# choose the desired scan numbers, using the python range function (the
second number is exclusive!)
scannums = range(6901, 6930)

# alternatively, you can also use a shell-glob pattern and do some rexexp
magic to obtain associated scan numbers
import glob, re
filenames = sorted(glob.glob('/daten/Raw/*_27-13_*.FITS'))
scannums = [re.search(r'20\d{6,6}_(\d{4,4})_.*', f).group(1) for f in
filenames]
```

```
for scannr in scannums:
    info = getScanInfo(scannr)
    print scannr
    subscanList = info.getSubscanNumList()
    for subscan in subscanList:
        try:
            reduceSubscan(scannr, subscan)
        except:
            pass
```

(In python it is important to keep indentation!)

### Options

There are a couple of options that can be passed to the `reduceSubscan` method.

| Keyword | Default | Description |
|---|---|---|
| febe | "*" | Process only data with a certain frontend-backend name (e.g., 'S60mm-XFFTS'), "*" means all |
| baseband | "*" | Process only data with specified baseband number (e.g., 1), "*" means all |
| fswFold | True | The pipeline does the frequency switch folding operation (you really want this and not let Class do it). See About Bandpass Ghosts. |
| regridMethod | "fft" | Which regrid algorithm to use for frequency shifting. The default ('fft') algorithm is fast and is the only which doesn't lower spectral resolution. **However, ringing effects might occur.** This is because the FFTS spectra are not fully-sampled in the strict sense of Shannon(-Nyquist)s sampling theorem. Ringing usually occurs when strong narrowband features are present. Other methods include 'gaussian' (for a convolution based approach) and scipy's regrid algorithms: 'linear', 'nearest', 'zero', 'slinear', 'quadratic', 'cubic'. |
| averMethod | "mean" | Allows to specifiy the averaging-method ('mean' or 'median', the latter might help with RFI) |
| skipDumps | (2, 0) | Allows to skip dumps at beginning/end of a subscan. At the beginning, the telescope sometimes is not yet fully on-source. |
| reverseArrayOrientation | False | The latter function additionally flips the data arrays for the following reduce operations, in case the IF flip was wrongly applied (usually happens only if testing new stuff). |
| shiftFFTSRefChannel | False | Before February 2011 the reference channel of the FFTS was wrong (in Fits header) and has to be shifted by +0.5 channels if the spectrum was not flipped, otherwise -0.5 channels. See also the News section. |
| autoLsrCorr | True | Per default the class pipeline attempts to correct the data for LSR Doppler shifts (due observer motion w.r.t. the LSR) if not already accounted for by hardware (LO tuning). If you really wanted a fixed-frequency setup (e.g., for system testing) you can disable automatic correction with this keyword. |

## Shell processing - CmdLinePipeline

Often one just wants to reprocess a bunch of Fitsfiles. For this, we made a small shell-(python-)script.

```
python CmdLinePipeline.py -h
```

```
usage: CmdLinePipeline.py [-h] [-c CLASSFILE] [--baseband BASEBAND]
                          [--febe FEBE] [--no-fsw-fold]
                          [--no-auto-lsr-correction]
                          [--regrid-method
{fft,gaussian,linear,nearest,zero,slinear,quadratic,cubic}]
                          [--averaging-method {mean,median}]
                          [--skip-dumps SKIPDUMPS SKIPDUMPS] [--spectral-
flip]
                          [--shift-ref-chan]
                          fitsnames [fitsnames ...]

Command line interface to Class pipeline

positional arguments:
  fitsnames             list of raw-fits files

optional arguments:
  -h, --help            show this help message and exit
  -c CLASSFILE, --classfile CLASSFILE
                        name of output class file (default
                        "<date>_<project>.100m") you can choose any name,
                        <date> and <project> are meta-variables and will be
                        replaced by what is found in the fits files
                        (potentially creating more than one class file)
  --baseband BASEBAND   Filter by baseband
  --febe FEBE           Filter by febe
  --no-fsw-fold         do fswitch fold in pipeline (rather than class)
  --no-auto-lsr-correction
                        do not automatically correct for LSR shifts in
fixed-
                        freq mode, default: False
  --regrid-method {fft,gaussian,linear,nearest,zero,slinear,quadratic,cubic}
                        which method to use for fswitch freq shifting
                        (default: fft) possible regridMethods are: fft,
                        linear, gaussian (the latter is recommended if you
                        experience ringing in the spectrum, linear and
                        gaussian decrease spectral resolution!)
  --averaging-method {mean,median}
                        averaging method (mean or median, default: mean)
  --skip-dumps SKIPDUMPS SKIPDUMPS
                        dumps to skip at beginning/end (tuple), default: 2 0
  --spectral-flip       additional spectral flip of the data arrays,
default:
                        False
```

```
  --shift-ref-chan      before February 2011 the reference channel of the
FFTS
                        had to beshifted by +0.5 channels if the spectrum
was
                        not IF flipped, otherwise -0.5, default: False
```

The meaning of the parameters is like in the Table above (just slightly different argument names). The only difference is the '-c CLASSFILE' option that allows to give a certain class name. Default is "<date>_<project>.100m" where the date and project code are automatically filled-in from what is found in the MBfits files (this can lead to multiple class files obviously).

Typical usage would be:

```
# put everything into one class file
python CmdLinePipeline.py -c '/homes/user/myclass.100m'
/daten/Raw/20141201*myprojectcode*XFFTS*

# put everything into separate class files (split by observing date and
project code)
python CmdLinePipeline.py -c '/homes/user/<date>_<project>.100m'
/daten/Raw/20141201*myprojectcode*XFFTS*

# process just one MBFITS file, and put class file(s) into current directory
python CmdLinePipeline.py /daten/Raw/20141211_2506_62-14_S13mm-XFFTS.FITS
```