Astronomical and Astrophysical Transactions (*AApTr*), 2021, Vol. 32, Issue 3, pp. 227–240,
ISSN 1055-6796, Photocopying permitted by license only, ©Cambridge Scientific Publishers

VIRAC maser data processing suite

J. Šteinbergs*, A. Aberfelds, M. Bleiders, I. Shmeld

Engineering Research Institute "Ventspils International Radio Astronomy Center", Ventspils University of Applied Sciences, Inženieru iela 101, Ventspils, LV-3601, Latvia

Received 21 April, 2021

Modern scientific research particularly radio astronomical spectroscopic observations cannot be imagined without appropriate software suite. That is necessary for two reasons: 1) data are in digital form and 2) data processing is a complex multi-step process. In this paper created in VIRAC data processing suite for a single radio telescope, space maser observations are presented. Software Defined Radio (SDR) backend data processing is described. Implementation of the frequency switching algorithm, acquisition of observation data and their format for storing are discussed. Multiple ways to display the observation results are highlighted.

Keywords: Python, GUI, maser, monitoring

1 Introduction

In the year 2017 Ventspils International Radio Astronomy Centre (VIRAC) team of astronomers and engineers started to work on the maser monitoring programme (Aberfelds et al., 2017). That included technical and scientific development. The technical development was creating the radio telescope backend and software package for data processing. The scientific development was the creation of source catalogue, developing mathematical background for data processing, creating observation planning method and studying the observed radio sources variability. Observations have been done with VIRAC 16 m and 32 m radio telescopes RT16 and RT32. Initially, both telescopes had Digital Base Band Converter (DBBC) (Tuccari, 2003) backend, recently SDR (Bleiders et al., 2020) based backend has been developed. Afterwards tool complex for single-dish radio telescope space maser observation data processing named *Maser Data Processing Suite* (MDPS). MDPS discussed in this paper was developed in VIRAC. It is open-source software, with user-friendly Graphical User Interface GUI distributed via GitHub repository `https://github.com/sklandrausis/Maser-Data-Processing-Suite`.

*Email: janis.steinbergs@venta.lv

There are many other software packages for single dish data processing like Common Astronomy Software Applications (CASA), GILDAS (it is used to reduce all data acquired with the IRAM 30M telescope and the NOrthern Extended Millimetre Array NOEMA), GBTIDL (is an interactive package for reduction and analysis of spectral line data taken with the Robert C. Byrd), but none of the existing software packages are compatible with VIRAC SDR based backend setup. For example maser data reducing software used for Effelsberg telescope is not publicly available and pipeline code is very tailored to Effelsberg, so it would be a major effort to adapt it to VIRAC needs. It is also tied to the Multi-Beam FITS Raw Data Format (MBFITS) format and if MBFITs format is not used, it would be better off with a complete re-write anyway (Information obtained in personal communication with Benjamin Winkel). VIRAC setup produces the specific format of data that these packages do not support. And these packages do not have monitoring module. Additionally frequency shifting algorithm is not a widespread maser calibration method, because of that MDPS is one of the first of this kind of software.

2 Overview of MDPS

MDPS currently allows us to 1) process SDR output, 2) display maser sources spectral components variability over time monitoring and 3) visualise data for publications. MDPS also uses several Python libraries. The most important of them are *PyQt5* (riverbankcomputing.com, 2020), *Astropy* (Robitaille et al., 2013) (Günther et al., 2018), *Matplotlib* (Hunter, 2007), *Numpy* (Oliphant, 2006), *H5py* (Collette, 2013), *SciPy* (Virtanen, 2020), *jplephem* (Rhodes, 2011) and *experimentsLogReader* (Šteinbergs, 2020). It also needs planetary ephemeris parameter file *de435.bsp* that can be required by `https://naif.jpl .nasa.gov/pub/naif/generic_kernels/spk/planets/de435.bsp.` The processing suite also uses two configuration files – Python library *Matplotlib* configuration file to modify plotting view, custom made configuration file that specifies used directories and hard-coded variables. The most important of them are 1) velocity line parameters, 2) signal cut values, 3) observed source parameters (Right ascension, Declination, epoch), 4) base frequencies of observed maser species and 5) station coordinations. Velocity line parameter is used to compute the local maximum of the spectrum and signal cut values are used to compute signal to noise ratio. The base frequencies of species are laboratory determined maser frequencies for specific chemical species and are used to compute the Doppler effect. Station coordination's are three geocentric values in metres (x,y,z) used to compute local standard of rest.

3 SDR backend output processing

3.1 SDR backend description

As backed on-the-shelf SDR Ettus Research USRP X300 equipped with TwinRX daughterboard is used (Bleiders et al., 2020). It is a multi-channel SDR transceiver that allows us to obtain complex sample rates and since USRP X300 is used together

with TwinRX front-end daughterboard, it allows dual-polarization spectroscopy (Bleiders et al., 2020). Optimal linearity receiver is found using calibration noise injection function (Bleiders et al., 2020). Since power values are measured with noise source on and off, it should not depend on input signal level that allows us to find optimum attenuation values for USRP unit, telescope receiver and IF signal chain itself (Bleiders et al., 2020).

3.2 Implementation of frequency switching algorithm

3.2.1 Mathematical implementation of frequency switching algorithm

Observations are done in a frequency-switching mode described by (Winkel et al., 2012) with four typically 15 seconds long integration stages. The first two stages are with noise diode turned off, the last two stages with noise diode turned on. These stages are marked as ref_{off}, sig_{off}, ref_{on} and sig_{on}. Definitions of *ref*, *sig* correspond to the shift of the local oscillator frequency to up and down from the calculated central frequency. Shifting local oscillator frequencies allow us to remove the bandpass dependence. Indexes *on* and *off* correspond to noise diode turned on and off respectively. Sequential SDR backend creates a file for each of these stages. Each of these files contains a matrix with three columns – the first is frequency, the second and third are amplitudes of left and the right polarisation. The system also generates a log file. It is parsed and necessary information for calibration is extracted by using Python library *experimentsLogReader*. The first step of telescope data processing is to rearrange data by shifting the centre-frequency component to the centre of the array after that calibration process can start. The calibration process is a multi-step process beginning with a conversion from telescope backend relative amplitude units to Kelvins. Thereafter the system temperature for noise diode turned off must be computed. This is done for both *sig* and *ref* stages. For the first, it is done with the help of formula 1. For the next one, the formula is similar only there is a $Pref$ instead of $Psig$.

$$Tsys_{off} = Tcal\frac{Psig_{on} + Psig_{off} - (Psig_{on} - \bar{} Psig_{off})}{2(Psig_{on} - \bar{} Psig_{off})} \tag{1}$$

Here $Tcal$ is the temperature of noise diode obtained from the observation log file. When observation log file is generated $Tcal$ value is written 3.820791 K for both telescope, that is temperature of noise diode coresponding the value of its generated signal. $Psig_{on}$ and $Psig_{off}$ is *sig* stage on and off amplitude. $(Psig_{on} - \bar{} Psig_{off})$ is an average value from the subtraction of $Psig_{on}$ and $Psig_{off}$. Next the calibrated amplitude for all four stages must be computed. Since computation for *sig* and *ref* stage is similar only for *sig* stage it will be shown. First $Tcal^{sig}_{off}$ for *off*-stage is computed using formula 2, thereafter $Tcal^{sig}_{on}$ for *on* stage using formula 3,

$$Tcal^{sig}_{off} = Tsys_{off}\frac{Psig_{off} - Pref_{off}}{Pref_{off}} \tag{2}$$

where $Psig_{off}$ is *sig* stage off amplitude.

$$Tcal^{sig}_{on} = (Tsys_{off} + Tcal)\frac{Psig_{on} - Pref_{on}}{Pref_{on}} \tag{3}$$

The next step is to compute average amplitude from their on and off stage values, that can be done by formula 4.

$$Tsig = (Tcal_{off}^{sig} + Tcal_{on}^{sig})/2 \tag{4}$$

After that data are shifted by shifting factor, computed by formula 5. In *sig* stage positive shifting factor is used in *ref* stage negative shifting factor is used. This is done by using the Python library *Numpy* function *roll*.

$$n_{shift} = \lfloor f_{shift}/f_{step} \rceil \tag{5}$$

where f_{step} is the distance between frequencies points and f_{shift} are computed by the formula 6.

$$f_{shift} = bw/df_{div} \tag{6}$$

where bw is bandwidth and df_{div} is division factor. Both values are obtained from the observation log. After that average of *sig* and *ref* stages are computed by using formula 7.

$$T = (Tsig + Tref)/2 \tag{7}$$

where *Tref* is calibrated *ref* stage amplitude obtained similar to *Tsig*. Finally, conversion from amplitude in Kelvins to amplitude in Janskys is done by formula 8.

$$S = \frac{T}{DPFUPoly_{val}(g_{el}, elevation)} \tag{8}$$

where S is amplitude in Janskys, T is calibrated amplitude, DPFU is degrees-per-flux-unit and $Poly_{val}(g_{el}, elevation)$ is evaluated ordinary polynomial, where the g_{el} is polynomial coefficients. Elevation, g_{el} and DPFU are obtained from separately performed calibration sessions, targeting well known continuum sources (Perley and Butler, 2013) and also are stored in log files. Polynomial is evaluated using Python library *Numpy* function *polyval*. The final part is to extract valuable part of data, that is done by computing two indexes. These indexes i and j are 1/4 and 3/4 of sampling points. Example of a frequency switching algorithm result screenshot is shown in Figure 1.

3.2.2 Practical details

SDR backend creates a file for each of calibration stages (ref_{off}, sig_{off}, ref_{on}, sig_{on}) for each scan. Data in these files are stored in American Standard Code for Information Interchange (ASCII) format. Each of these files contains a matrix with three columns – the first is frequency, the second and third are amplitudes of left and the right polarisation, no header information. These files are read using Python library *Numpy* function *loadtxt*. Before reading the data files they are sorted for each scan and checked to confirm if every scan has all four files, if not, the scan is deleted and the user is warned about that. SDR backend file naming scheme is this $\langle source \rangle_f\langle frequency \rangle_\langle station$

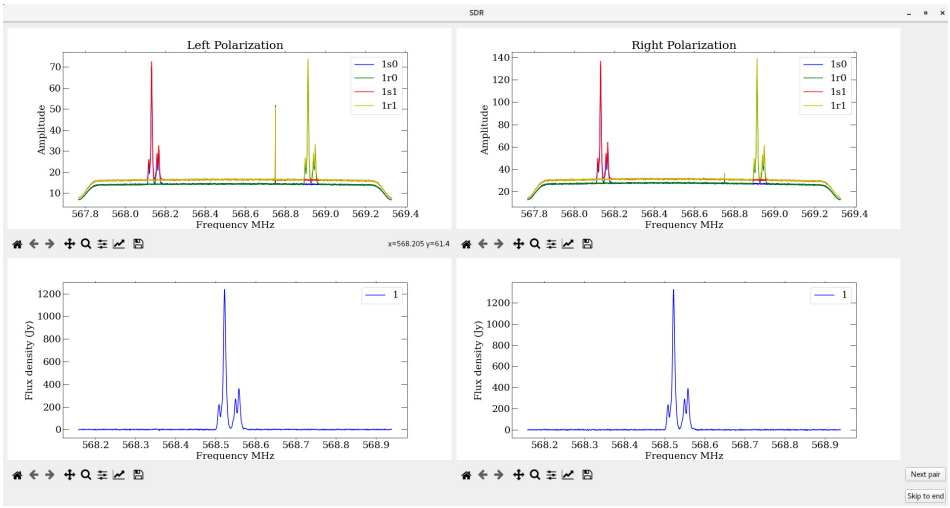


Figure 1 Upper panel amplitude in four stages as a function of frequency. Bottom panel calibrated amplitude in Janskys as a function from frequency. Source G109.871+2.114/Cepheus A was selected as example.

⟩_⟨*iteration*⟩_*no*⟨*nOandstage*⟩.*dat*, where *nOandstage* is scan number and calibration stage index, example *001r0*. All scan numbers and calibrator stage index contain two parts scan number in example *001* and calibration stage index *r0*. Calibration stage index is string that contains two parts identification for *sig* and *ref* stages – letter *r* and *s* and identification for *on* and *off* stages numbers *1* and *0*. These files are read as *Numpy* arrays and all mathematical computations mentioned in 3.2.1 are done using matrix operation.

3.3 Computation of Doppler effect

After amplitudes are calibrated, the next step of data processing is to compute the Doppler effect and transform data from the frequency domain to the velocity domain, and this is done using formula 9.

$$v = v_r - (\frac{f_o}{f_b} - 1)c \tag{9}$$

where f_o is observed frequency, f_b is laboratory determined maser frequency, c is the speed of light and v_r is the velocity of the receiver.

The observed frequency is frequency from data file plus local oscillator frequency. The velocity of the receiver in Local Standard of Rest frame is the sum of Sun apex velocity, Earth orbital velocity and Earth rotation.

3.4 Observation data quality estimation

Data quality is determined by signal to noise ratio. This ratio is computed in the velocity domain. To extract a signal from data MDPS configuration file has section

cuts, that contain velocities ranges where signal is located. This information is used to compute indexes of ranges for a signal. Knowing where a signal is in data, it is possible to know where noise is in data. Then the signal to noise ratio is computed by the maximum value of signal divided by 3 multiply by standard deviation of the noise.

An additional option to estimate the quality of observation is to see the system temperature over time. In a normal situation, it will be stable. The result screenshot example is shown in Figure 2

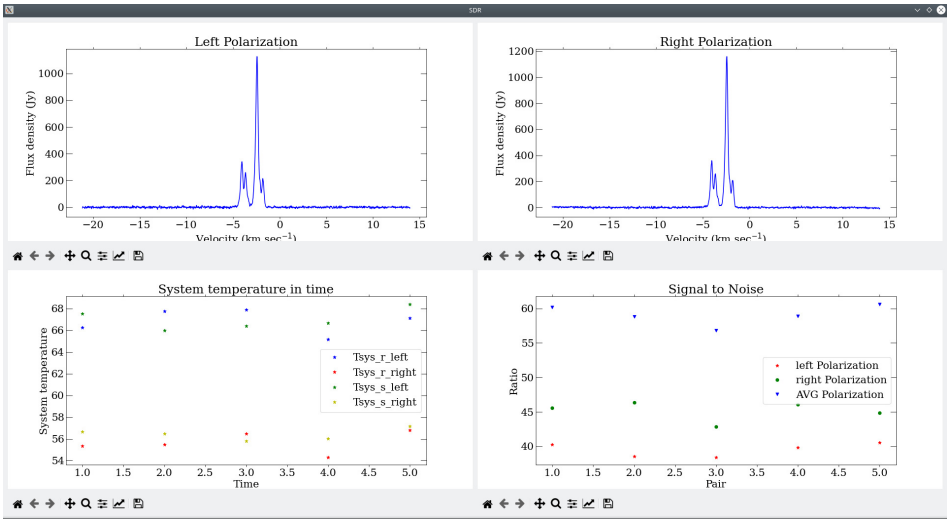


Figure 2 The upper panel amplitude as a function from velocity. The Bottom panel left side system temperature as a function of time, right side signal to noise ratio as a function of time. Source G109.871+2.114/Cepheus A was selected as an example.

3.5 Data normalisation and monitoring values extraction

After transformation to velocity domain scans of observation are done and aligned so that the maximum value index is the same for all scans, then all scans are averaged to avoid velocity drift during the observation time. When a single spectrum is created from all scans in one observation, data must be normalised. It is necessary because during observations the effect from system instabilities will correspond to no flat noise floor in the spectrum. It is done by computing ordinary polynomial from the noise part of the spectrum and this polynomial is subtracted from the spectrum. By default, MDPS use polynomial order 3. An example screenshot is shown in Figure 3.

After data are normalised, amplitudes for maser components can be obtained. To do that maser component velocity values are used. First indexes for component velocities are computed, after that ranges of local maximum areas are constructed. This is done by using previously computed indexes for component velocities and index range for local maximum are added and subtracted to these indexes. For ranges of local maximum areas, a maximum value for amplitude is computed and stored

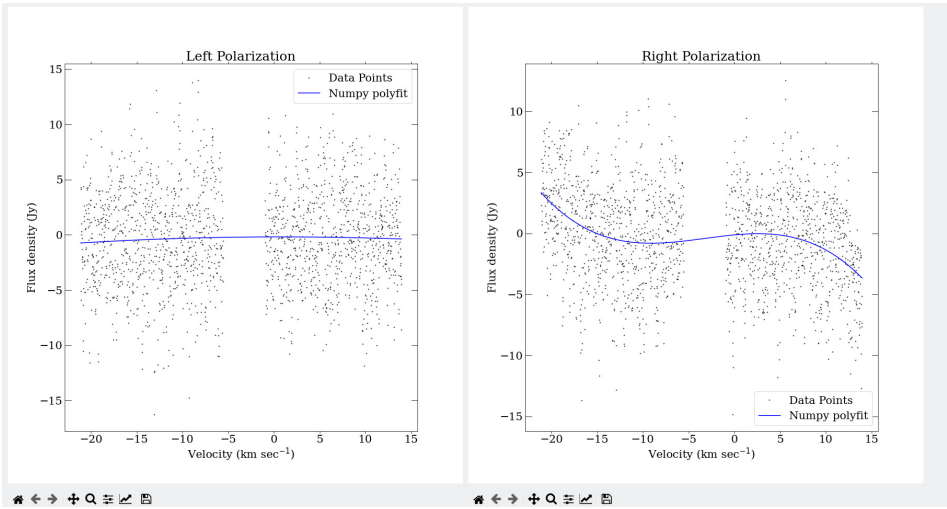


Figure 3 Black dots data points of signal noise part, blue line polynomial. Source G109.871+2.114/Cepheus A was selected as an example.

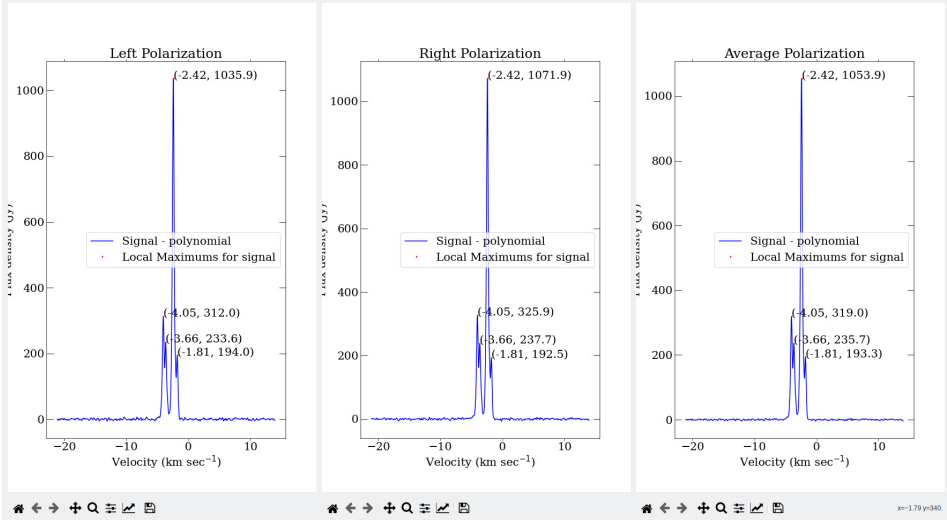


Figure 4 Blue line spectrum, red dots amplitude values for maser components. Source G109.871+2.114/Cepheus A was selected as an example.

for monitoring. The result for found amplitude values for maser components result screenshot is shown in Figure 4.

3.6 Output data products of SDR backend data processing and their formats

After processing SDR backend data MDPS creates two outputs 1) calibrated spectrum data and 2) data for spectra variability with time monitoring. For calibrated spectrum data file format the following specifications were selected: 1) it must have read and written a library in most popular programming languages (output files can be distributed to other researchers and they can use any tools they like to process these files); 2) it should be possible to store data for each data processing stage in one file (it is necessary to avoid complicated file naming schemes and that allow reprocessing data from any data processing stage). To store and process astronomical data many data formats are used. In this paper the authors will only discuss these data file formats (CSV, FITS (Wells et al., 1979), HDF5 (Folk et al., 2011), NPY (community, 2020), MS (Kemball and Wieringa, 2000)). They are given in Table 1.

Table 1 Overview of data files used in astronomy.

File format	Programming languages supporting format	Comments
CSV	Most of languages	Cannot store multiple matrices in one file
FITS	C, Fortran, C++, C#, Python, IDL, Java, JavaScript, Julia, TCL, Perl, MatLab, Mathematica, R, Go, Swift (McGlynn, 2020)	Primarily used for image transport and archiving
HDF5	C, C++, Java, Python, C#, Fortran, R, S-Lang, MatLab, Mathematica (hdfgroup.org/, 2020))	Can be used for storing complex data structures
NPY	Only Python	*Numpy* matrix output format. Cannot be used to store multiple matrices and meta data
MS	C++, Python	Primarily used for image transport

We see that both criteria are satisfied by only two data formats, FITS and HDF5. Although FITS file format has the support of more programming languages, we favour HDF5, because of advantages that have HDF5. These advantages are as improved I/O speed, compression (Price et al., 2015) and HDF5 data format is used in new modern telescopes like LOFAR, CCAT and MeerKat (Price et al., 2015) and will be used in the future telescope SKA (Comrie et al., 2020) software. The Output file contains tables *amplitude*, *amplitude_ corrected*, *amplitude_ corrected_ not_ smooht*, *specie* and *system_ temperature*. Table *amplitude* is the result of frequency switching algorithm, table *amplitude_ corrected_ not_ smooht* contains normalises data, *amplitude_ corrected*

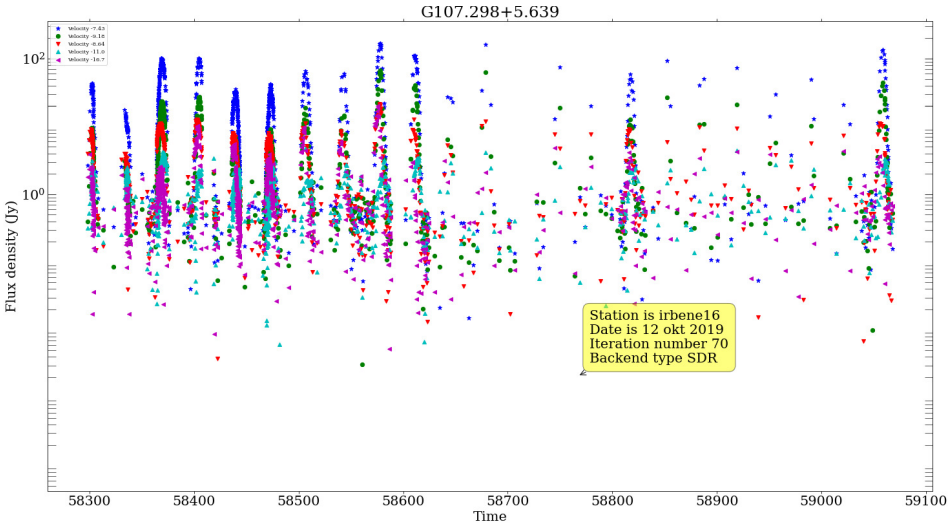


Figure 5 G107.298+5.639 components amplitude as a function of time. Each component has a different colour.

smoothed data table specie a specie of observed source and table *system_ temperature* contains system temperature as a function of time.

For variability monitoring data the following specifications were selected: 1) it must allow monitoring meta-data to change over time; 2) it must have reading and writing library in most popular programming languages. Therefore JavaScript Object Notation (JSON) was chosen as data format.

4 Variability monitoring

4.1 Single source monitoring

This consists of two components: single source monitoring and multiple source calibrator line monitoring. The monitoring result is a maser spectrum components amplitude as a function of time. A simple example can be seen in Figure 5.

An additional option is to compute power as a function of periods, as seen in Figure 6. Power as a function of periods is obtained by computing Lomb–Scargle periodogram as described in (VanderPlas and Ivezic, 2015). To compute periodogram Python library *Astropy*, modules time series function *LombScargle* was used. Monitoring module also can compute amplitude changes not only for specific maser velocity component, but also can compute amplitude change as a function in time for all spectra. That is done by computing triangulation between the velocity of maser and observed time. After that contour of the created triangular grid is drawn, contour colours correspond to observed fluxes. The result can be seen in Figure 7.

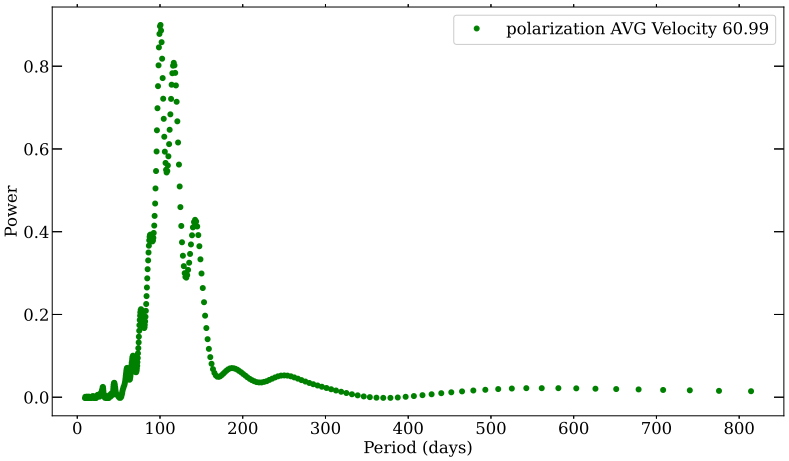


Figure 6 G33.641-0.228 power of velocity 60.99 component as a function of periods.

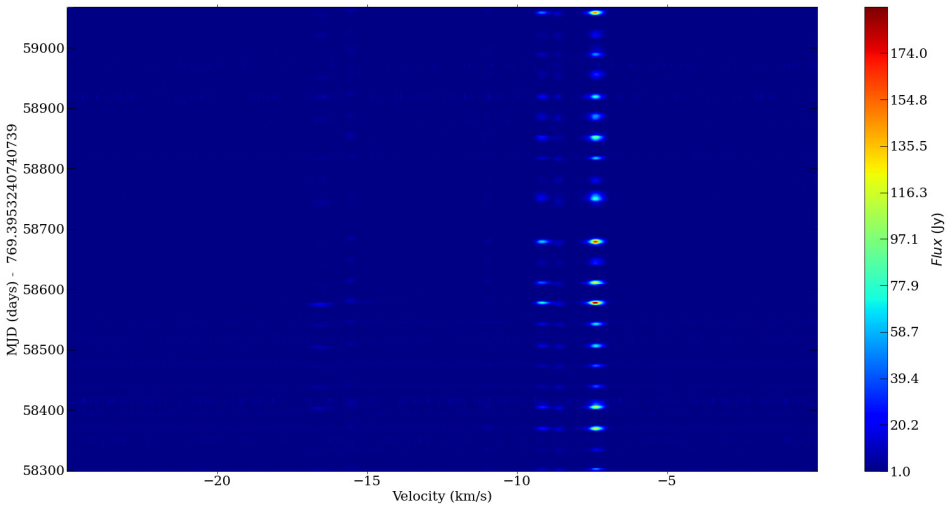


Figure 7 G107.298+5.639 Amplitude change as a function in time for all spectra (spectrogram).

4.2 Multiple source calibrator line monitoring

From all sources included in the maser monitoring programme, stable velocity components are selected. Currently these G32.744-0.076 (30.49, 39.18), G49.490-0.388/W51 (59.29), G59.783+0.065 (19.2), G69.540-0.976/ON1 (14.64), G188.95+0.89/S252 (10.84), G111.542+0.777/NGC7538 (-58.04), G133.947+1.064/W3(OH)) (-44.6) components are selected. This kind of monitoring allows us to detect instrumental errors. Because selected velocity components have different amplitude values, before plotting data must be normalised, and this is done by dividing each selected velocity

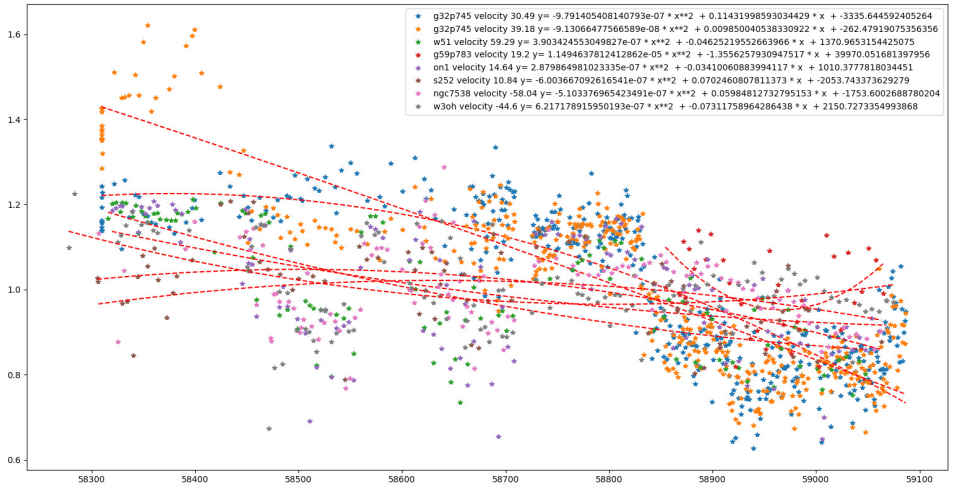


Figure 8 Multiple source calibrator line monitoring.

components with an average for that component. For each of these components, a second-order polynomial is computed. The result can be seen in Figure 8.

5 Performance testing

Running Python scripts with option *-m cProfile* print in terminal profile info (function calls, primitive calls, execution time in seconds) or importing Python library *cProfile* to script it will allow us to get profile info from any part of the script. Figure 9 shows SDR backend output processing execution with different number of scans, but all observations had 4096 numbers of points.

With reference to Figure 9 we can see that execution time is between functions O(logn) and O(nlogn) that means that complexity of SDR backend data processing algorithms is O(n).

Similar analyses are shown spectrogramm computation, results of that can be seen in Figure 10, but execution speed of spectrogramm computation is determined by numbers of observation in monitoring and numbers of spectral components.

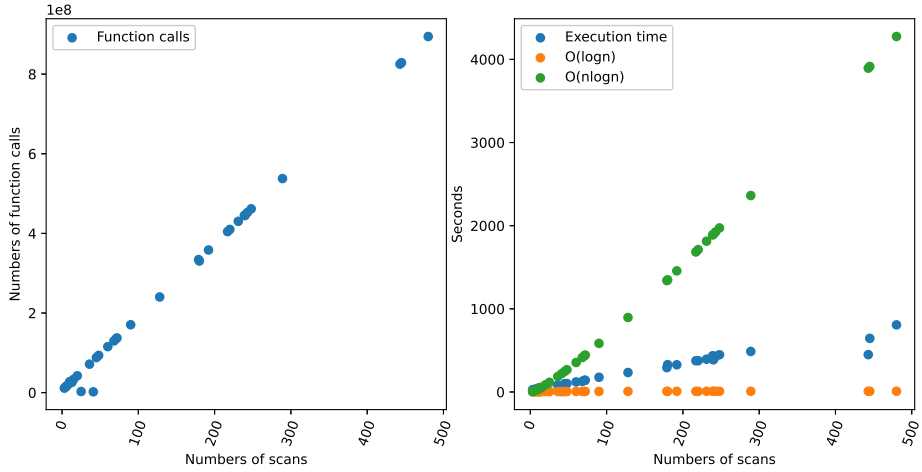


Figure 9 X-axis, numbers of scans. Y-axis: left side, numbers of function calls; right side, seconds of execution. On the right side, functions O(logn) and O(nlogn) are also shown.

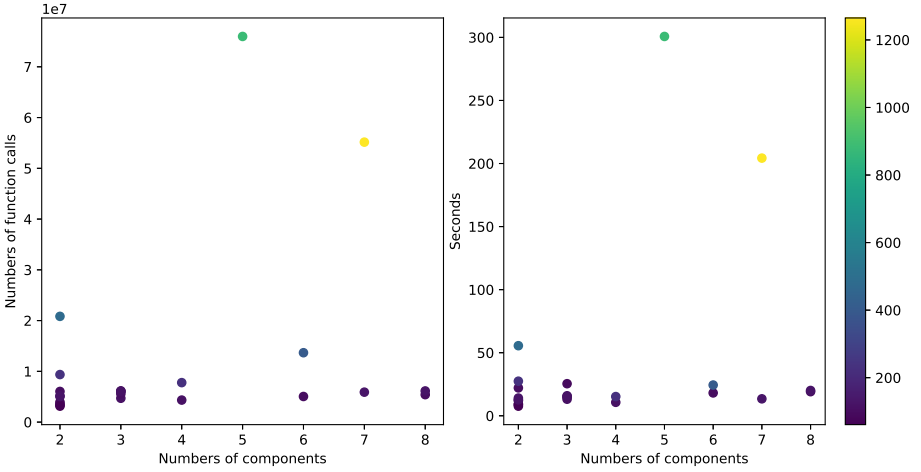


Figure 10 X-axis, numbers of spectral components. Y-axis: left side, numbers of function calls; right side, seconds of execution. Colours scheme is based on numbers of observation in monitoring.

6 Conclusions

This paper presents the developed maser line monitoring processing suite. It enables process SDR output, display monitoring and visualises data for publications and presentation. The suite was entirely written in programming language Python. For data storing modern and efficient technologies were used, like HDF5. Implementation of the frequency switching algorithm was described in detail. Variability monitoring

module was described and different ways to display monitoring results were shown. Also, data formats for storing the processed data were discussed.

Acknowledgements

This work was supported by the Latvian Council of Science Project "Research of Galactic Masers" No lzp-2018/1-0291.

References

Aberfelds, A., Shmeld, I., and Berzins, K. (2017). Long term 6.7 ghz methanol maser monitoring program. *Proceedings of the International Astronomical Union*, 13(S336):277 – 278.

Bleiders, M., Antyufeyev, O., Patoka, O., Orbidans, A., Aberfelds, A., Steinbergs, J., Bezrukovs, V., and Shmeld, I. (2020). Spectral line registration back-end based on usrp x300 software defined radio. *Journal of Astronomical Instrumentation*, 9(2).

Collette, A. (2013). *Python and HDF5: unlocking scientific data.* " O'Reilly Media, Inc.".

community, T. S. (2020). numpy.load.

Comrie, A., Pińska, A., Simmonds, R., and Taylor, A. (2020). Development and application of an hdf5 schema for ska-scale image cube visualization. *Astronomy and Computing*, page 100389.

Folk, M., Heber, G., Koziol, Q., Pourmal, E., and Robinson, D. (2011). An overview of the hdf5 technology suite and its applications. In *Proceedings of the EDBT/ICDT 2011 Workshop on Array Databases*, pages 36–47.

Günther, H., Lim, P., Crawford, S., Conseil, S., Shupe, D., Craig, M., Dencheva, N., Ginsburg, A., VanderPlas, J., Bradley, L., et al. (2018). The astropy project: Building an inclusive, open-science project and status of the v2. 0 core package. *arXiv preprint arXiv:1801.02634*.

hdfgroup.org/ (2020). Software using hdf5: Descriptions.

Hunter, J. (2007). Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3):90–95.

Kemball, A. and Wieringa, M. (2000). Measurementset definition version 2.0. *URL: http://casa. nrao. edu/Memos/229. html*.

McGlynn, T. (2020). Fits i/o libraries.

Oliphant, T. E. (2006). *A guide to NumPy*, volume 1. Trelgol Publishing USA.

Perley, R. and Butler, B. (2013). An accurate flux density scale from 1 to 50 ghz. *The Astrophysical Journal Supplement Series*, 204(2):19.

Price, D., Barsdell, B., and Greenhill, L. (2015). Hdfits: Porting the fits data model to hdf5. *Astronomy and Computing*, 12:212–220.

Rhodes, B. C. (2011). Pyephem: astronomical ephemeris for python. *ascl*, pages ascl–1112.

riverbankcomputing.com (2020). What is pyqt?

Robitaille, T., Tollerud, E., Greenfield, P., Droettboom, M., Bray, E., Aldcroft, T., Davis, M., Ginsburg, A., Price-Whelan, A., Kerzendorf, W., et al. (2013). Astropy: A community python package for astronomy. *Astronomy & Astrophysics*, 558:A33.

Šteinbergs, J. (2020). experimentlogreader.

Tuccari, G. (2003). Development of a digital base band converter (dbbc): Basic elements and preliminary results. In *New technologies in VLBI*, volume 306, page 177.

VanderPlas, J. and Ivezic, Ž. (2015). Periodograms for multiband astronomical time series. *The Astrophysical Journal*, 812(1):18.

Virtanen, P. e. a. (2020). SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272.

Wells, D., Greisen, E., and Harten, R. (1979). Fits-a flexible image transport system. In *Image Processing in Astronomy*, page 445.

Winkel, B., Kraus, A., and Bach, U. (2012). Unbiased flux calibration methods for spectral-line radio observations. *Astronomy & Astrophysics*, 540:A140.