algorithms

MDPI

Article

Hyperspectral Python: HypPy

Wim Bakker *, Frank van Ruitenbeek, Harald van der Werff *, Christoph Hecker, Arjan Dijkstra and Freek van der Meer

Faculty ITC, University of Twente, 7522 NH Enschede, The Netherlands; f.j.a.vanruitenbeek@utwente.nl (F.v.R.); c.a.hecker@utwente.nl (C.H.); a.h.dijkstra@utwente.nl (A.D.); f.d.vandermeer@utwente.nl (F.v.d.M.)
* Correspondence: w.h.bakker@utwente.nl (W.B.); h.m.a.vanderwerff@utwente.nl (H.v.d.W.)

Abstract: This paper describes the design, implementation, and usage of a Python package called Hyperspectral Python (HypPy). Proprietary software for processing hyperspectral images is expensive, and tools developed using these packages cannot be freely distributed. The idea of HypPy is to be able to process hyperspectral images using free and open-source software. HypPy was developed using Python and relies on the array-processing capabilities of packages like NumPy and SciPy. HypPy was designed with practical imaging spectrometry in mind and has implemented a number of novel ideas. To name a few of these ideas, HypPy has BandMath and SpectralMath tools for processing images and spectra using Python statements, can process spectral libraries as if they were images, and can address bands by wavelength rather than band number. We expect HypPy to be beneficial for research, education, and projects using hyperspectral data because it is flexible and versatile.

Keywords: hyperspectral imaging; image processing; Python; minerals

check for updates

Citation: Bakker, W.; van Ruitenbeek, F.; van der Werff, H.; Hecker, C.; Dijkstra, A.; van der Meer, F. Hyperspectral Python: HypPy. *Algorithms* **2024**, *17*, 337. https://doi.org/10.3390/a17080337

Academic Editors: Laura Antonelli and Lucia Maddalena

Received: 27 June 2024
Revised: 22 July 2024
Accepted: 30 July 2024
Published: 1 August 2024

1. Introduction

Image processing packages have been around since the creation of the first computers. However, at the start of this century, not many packages were capable of handling hyperspectral image data. Hyperspectral images tend to be huge, and standard operations such as image classification are unfit for handling the high dimensionality of hyperspectral data. Traditionally, hyperspectral images are processed using proprietary packages such as ENVI (ENVI and IDL are trademarks of NV5 Geospatial Solutions, Inc.) (Environment for Visualizing Images) [1] and others. However, with the advent of free open-source software [2], the need was felt to be able to use and process hyperspectral images independently of proprietary packages.

With packages like numpy [3], scipy [4], and matplotlib [5], the Python language [6] has become a data-processing and prototyping environment, rivaling systems (proprietary and free) such as MATLAB (MatLab is a registered trademark of The MathWorks, Inc.), IDL (ENVI and IDL are trademarks of NV5 Geospatial Solutions, Inc.), Octave (Octave is a trademark of Octave Technologies, Inc.), and R.

Python, in addition the aforementioned packages numpy and scipy, is free and open-source. Users can access the source code, make modifications, and distribute their own versions. In principle, any code written in Python can be distributed freely.

We use Python for the implementation of the package Hyperspectral Python, HypPy (pronounced [hip-ee], like the subculture in the 1960s), which is a software package for reading and writing, visualizing, processing, and analyzing hyperspectral data.

HypPy started as a number of image classes and factory functions enabling the reading and writing of ENVI format images. The creation of more tools in HypPy gained momentum when our research group became involved in the Mars Planetary Mapping Pilot Project [7] in 2009 and 2010. Subsequently, various general tools and specific tools for handling Mars hyperspectral data were written. At the same time, HypPy offered a platform for implementing experimental ideas. Gradually, the package was extended to what it is today. This paper describes the design considerations and implementation of HypPy providing a number of examples.

2. Design and Implementation

In this section we aim to show that HypPy has a set of features, which sets it apart from other packages and makes it a versatile and flexible tool for processing hyperspectral data.

HypPy has the following features:

- HypPy supports three levels of interfacing: a graphical user interface (GUI), a command-line interface (CLI), and a programming interface (API). The user interfaces of HypPy can be accessed from one top-level menu.
- The top-level menu is configurable and can be adapted for different purposes.
- HypPy has viewers for hyperspectral images and spectral libraries. These viewers have simple interfaces yet offer sufficient functionality for exploring the quality and information content of the data.
- HypPy offers a read and write capability for ENVI data files.
- Hyperspectral images tend to be huge, yet HypPy is memory-friendly.
- HypPy supports ENVI's three native image formats: BSQ, BIL, and BIP. Internally, HypPy translates these to a BIP format, relieving the programmer of the hassle of having to deal with three different formats.
- HypPy offers functionality for working with wavelengths instead of band numbers. Band numbers have no physical meaning, whereas wavelengths do.
- Raw image data may not have bands in the order of increasing wavelengths. Therefore, HypPy is able to sort wavelengths in the image on-the-fly.
- HypPy is able to use the bad band list (bbl) and skip bands that are marked as bad.
- HypPy can process ENVI spectral libraries as if they were ENVI images.
- If necessary, spectra are resampled on-the-fly. This eliminates the need to spectrally resample before combining spectra from different sources.
- HypPy is free and open-source software, which makes it an ideal tool for education. Students can pick up the package for free and use it at home as well.

In the following sections, the points mentioned above will be discussed in more detail.

2.1. The Usage Levels: GUI, CLI, and API

Image processing packages support different types of interfaces to the software. For instance, the Khoros package [8] support a GUI, CLI, and API. On top of that, Khoros offers an interface for visual programming, which is called Cantata [9]. Building a visual programming environment is complex and is not within the scope of HypPy. However, similar to Khoros, HypPy offers a GUI, CLI, and API.

The advantage of having these three levels is that it offers flexibility. GUIs make it easy to use the tools, even for beginners. From the interface, it is clear which inputs and outputs a certain tool may need. In addition to that, a GUI can be used to invoke multiple tasks at once. In this way, two or more programs may be run in parallel, thus enabling the use of multi-processing capabilities of the computer without the need for parallel coding.

The CLI can be used to run tools on the command line. However, the real power of the CLI is that it can be used to create scripts that run a sequence of tools in order to run a complete processing chain. Furthermore, scripting can also be used for time-consuming tasks that run for a long time or automate the processing of large sets of images.

Most of the functionality of HypPy can also be imported into other Python programs using the API.

The GUI of a HypPy tool consists of a separate module, which imports the HypPy function, creates the GUI, and passes the parameters to the HypPy function. One module contains the GUI, while another module contains the API and the CLI. For instance, the edgy.py module contains the functions for hyperspectral edge filtering plus the command-line interface. The GUI is invoked using the tkEdgy.py module. In this case, the three levels are as follows:

- GUI program: tkEdgy.py;
- CLI program: edgy.pyl
- API function: edgy().

The HypPy top-level menu can be used to start the GUI program tkEdgy.py. The Edgy filter can be found as a 'Spatial–Spectral Edge Filter' in the 'Filter' menu.

2.1.1. GUI

Most HypPy tools are split into two programs, one for the GUI and one for the API and the CLI. Most GUIs are simple interfaces for setting input and output parameters, which are passed to the API. An example of a GUI can be found in Section 2.4.1, where the Band Math Tool is discussed.

The user interfaces for the GUI processing tools are created using Tkinter, the GUI (graphical user interface) package that is part of the official Python distribution. Tkinter is a thin object-oriented layer on top of Tcl/Tk [10]. There are many packages that can be used to develop GUIs in Python. For instance, wxPython is a more extensive package (e.g., a widget set) than Tkinter. However, the Tkinter module (Tk interface) is the standard Python interface to the Tk GUI toolkit [11] and was chosen for HypPy for its simplicity and availability. All the names of the HypPy tools start with 'tk' to indicate that these have a GUI.

2.1.2. CLI and Scripting

The command-line interface is useful for running processes at the command prompt. However, the most important reason to have a CLI is to be able to run a processing chain from a script. Scripts are convenient for repeating a set of commands in an iterative manner. In this way, a potentially large set of images can be processed in exactly the same way. Running scripts reduces errors and offers the possibility to run commands that require a lot of processing time overnight.

HypPy adheres to the conventions used to supply command-line options and input and output on the command line. For that purpose. it uses the Python standard module called argparse, which offers a convenient way to build command-line interfaces that can be used in Unix as well as Windows shells.

2.1.3. The API

An application programming interface (API) is a software interface by which one piece of software offers functionality to other software. In Python, functionality from one program can be imported into another program by making use of the import statement. An important part of the API is the API specification, which describes the functions and data structures that are needed to make use of the available functionality.

Currently, a dedicated API manual is not present in HypPy, but essential classes like Image and Spectrum are covered in the Programmers' Manual and the Spectral Math Manual, respectively. Additionally, many HypPy functions are described in the HypPy Scripting Manual (CLI) and HypPy User Manual (GUI), as the CLI and GUI often act as a thin shell around these functions.

Because data objects are an important part of the API, HypPyś header and image objects are discussed below; see Section 2.2.

2.1.4. The Top-Level Menu

The top-level menu of HypPy can be configured using the menu configuration file called hyppymenu.cfg. Functions can be Python scripts (typically the HypPy GUI programs), PDF or HTML files, or complete URLs. In this way, the top menu can be used to start scripts as well as present documents such as manuals or web pages.

The tools are executed as separate processes; therefore, multiple HypPy tools can be started and executed in parallel at the same time. Figure 1 shows the top menu of HypPy.

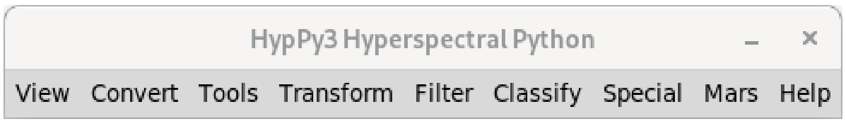


Figure 1. The top-level menu of HypPy.

Python scripts are executed by Python as subprocesses. The PDF, HTML, or URL is delegated to the default reader of the system, which will start the PDF reader or web browser on the system.

2.2. Image Formats

HypPy has adopted the ENVI formats as its native formats. The ENVI image and ENVI spectral library formats are binary files with an accompanying text header file. ENVI supports three different types of interleaving in its native format: band sequential (BSQ), band-interleaved by pixel (BIP), and band-interleaved by line (BIL).

HypPy supports all three ENVI formats: BSQ, BIP, and BIL. In addition, HypPy can map an ENVI spectral library onto an image. The spectral library is then represented as a y by x by z image, in which y is the number of spectra stored in the spectral library, x is always 1 (the spectral library is visible as an image with just one column of pixels), and z is the number of bands in the spectral library. The advantage is that spectral libraries can be viewed just like spectral images. On top of that, a processing chain meant for spectral images can be tested on a spectral library first to see what effect the processing chain will have on the various minerals present in the spectral library.

For consistency in coding, single-band images are mapped onto three dimensions as well, in which case the band dimension is always 1. There is one exception to this rule: classification images are mapped as two-dimensional one-band images. Classification images have a different use and semantics than hyperspectral data, and for this reason, classification images are treated in a different way in HypPy.

Internally, HypPy maps all three formats, BSQ, BIP, and BIL, onto an array with a BIP structure, thus relieving the programmer of having to deal with three different formats.

2.2.1. Header and Image Objects

HypPy's header module is used to read and write ENVI header files. ENVI images and spectral libraries typically consist of two files, the data and the header. The data files contain raw binary data. The header files contain human-readable text.

Inside HypPy, the ENVI header is represented as a Python object of the class Header. Objects of this class contain attributes that reflect the attributes as they are found in the ENVI header files. In the Python code, there are a few differences. First, the names of the attributes in the ENVI header can contain spaces, which is inconvenient in a programming language like Python because variable names are not allowed to contain spaces. A choice was made to replace the spaces in ENVI attribute names with an underscore. For instance, the 'data-type' attribute in the ENVI file header will be renamed to 'data-type' in Python. Second, some of the values inside the ENVI header may change their appearance in Python. For instance, IDL records, as they appear inside ENVI headers, such as values of the wavelength attribute, are changed into the standard Python list data type. The translation between IDL records and Python lists is handled by HypPy internally.

HypPy supports all IDL/ENVI raw binary data types from 8-bit unsigned integers to 64-bit signed integers and from 32-bit single floating point numbers to 64-bit double-precision complex numbers with a real-imaginary pair of double precision. Furthermore, through the use of the 'byte order' field name, HypPy supports both big-endian and little-endian byte order files.

HypPy has two factory functions in its image module, the Open() function for opening existing images, and the New() function for setting up new (empty) images.

The factory function Open() returns one of the image objects of class ImageBIP, ImageBIL, ImageBSQ, Classification, or ImageSL (a spectral library mapped as an image). Images accessed using the Open() function can only be read, not written. To protect existing images, these are always opened as read-only.

The factory function New() returns one of the image objects of class ImageBIP, ImageBIL, ImageBSQ, or Classification. In addition to these classes, the class ImageSL is also supported, which is an image representation of an ENVI spectral library.

2.2.2. Memory Map

HypPy uses Numpy's memory map, 'numpy.memmap', to map binary data files onto arrays. Memory-mapped files are used to access large files on a disk without having to read the entire file into memory first. NumPy memmap objects are array-like objects; they can be manipulated in the same way as numpy arrays. Because, in HypPy the choice was made to have the BIP structure for the internal array, the axes of the two other formats, BSQ and BIL, must be swapped to look like BIP in the returned memory map. This is achieved by calling the transpose function of the memory map. BSQ is mapped using a transpose(1, 2, 0). BIL is mapped using a transpose(0, 2, 1). The BIP format does not need to be transposed.

One would think that using the transpose function would create a copy of the data with the axes of the data swapped in a different way. However, this is not the case because, for certain operations, numpy supports what is called a 'view' on the data. A view creates a reference to an existing array, with the required reorganization of the data added to the metadata. This has two advantages: first, the view does not create a copy of the data in memory, and second, when the view is opened in read–write, any modifications to the view will be reflected immediately in the original data.

Furthermore, endianness and byte order can be changed by creating a view. The three formats (BSQ, BIP, and BIL), endianness (little endian and big endian), and data type (from 8-bit unsigned integers to 64-bit floating points, etc.) can be handled by creating a view of the data on file. For the input and output image data, no copy needs to reside in the memory of the computer. In such a way, HypPy can handle large files that would normally not fit in memory.

The following functions are applied to create views:

- Creating a memory map:
  numpy.memmap(fname, mode = 'r', shape = shape, dtype = data_type);
- Changing the byte order: method newbyteorder();
- Swapping the axes from BSQ to BIP: method transpose(1, 2, 0);
- Swapping the axes from BIL to BIP: method transpose(0, 2, 1).

Unfortunately, changing the data type of an input file creates a copy in memory, not a view:

- Change the data type of the input: method astype(dtype).

However, there are a few caveats in the use of numpy memory maps. Not all operations and functions that can be used on numpy arrays can be used on memory maps, and only a limited number of operations or functions on a memory map are capable of returning a view.

Additionally, the use of a memory map itself has disadvantages as well. Setting up a memory map uses a bit of extra memory, and all the reorganization of a view must be executed on-the-fly whenever the data on file are accessed. In this way, the memory map is less efficient than making an array copy in memory.

The advantages of a memory map are obvious, it enables the handling of large datasets that cannot possibly fit in the physical memory of the computer. Furthermore, internally, the data are mapped on the same BIP structure all the time; in a program, there is no need to keep track of the real interleave format of the data file. All this is handled transparently by the memory map and the views thereof.

2.2.3. Spectral Library Formats

HypPy supports two basic types of spectral libraries: ENVI spectral libraries and ASCII or text-based spectral libraries. In ENVI spectral libraries, a major drawback is that the list of wavelengths has to be the same for all the spectra in the spectral library. Therefore, the number of bands and the central wavelengths of all the spectra must be the same for all included spectra. An advantage of this format is that it is a binary format and can be addressed like an image, which offers the possibility to process spectral libraries in the same manner as spectral images. This can be used to test processing chains on spectral libraries before running them on images. A process for detecting certain minerals can be tested on the USGS spectral library [12], for instance, before applying it to an image. Problems with false positives of certain minerals can be caught in the early stages of developing a processing chain.

On the other hand, the ASCII spectral libraries offer more flexibility. Text files can be imported into other programs, and the spectral ranges of the individual spectra do not need to be all the same. A complication is that spectra must be resampled before use. HypPy resamples spectra on-the-fly, using the interp1d() function from the scipy.interpolate module to unite both sets of wavelengths. An intermediate step of resampling spectra is not needed.

2.2.4. Bad Band Lists

The ENVI image format supports the notion of what is called a bad band list (BBL), indicated in an ENVI header file by the keyword BBL. There must be as many items in the BBL as there are bands in the image. Sequentially, the bands are marked as bad or good. A zero (0) for a particular band signifies that the data in this band should not be trusted because of a sensor failure or excessive noise. A one (1) for a band signifies valid data. HypPy can take into account the bad band list, such that the bad bands do not show up in array data. In most HypPy tools, this can be indicated by setting the use_bbl option to true. If this option is set to false, then all of the data are visible in the array, including the bad bands. Such a virtual image, on which the use_bbl is set to true, may have fewer bands than the original data file.

Internally, this is handled by an index-to-index array, which is used as an indexing array to access multiple elements at once. The disadvantage is that handling such virtual images is slower than using the full image. Another disadvantage is that the band numbers in the virtual image are not the same as the band numbers in the original image. However, one of the design ideas of HypPy is to use the wavelength rather than the band number to address a spectral band of an image (see next section).

2.2.5. Sorting Wavelengths

Often, sensors consist of a number of detector banks, each covering a specific wavelength range. The data from these detectors may have overlapping wavelength ranges or may even appear in an unexpected order. For instance, the raw data from the OMEGA instrument of the European Mars Express have the data from the SWIR1 (0.93 µm to 2.69 µm) and SWIR2 (2.53 µm to 5.09 µm) detector banks before the data from the VNIR detector (0.36 µm to 1.07 µm) [13]. Another well-known hyperspectral instrument, AVIRIS, consists of four spectrometers with overlapping ranges [14]. Note that redundant bands in the spectral overlap of the AVIRIS spectrometers can also be removed or put in the bad band list; see, for instance, [15].

Obviously, the switched order and overlapping ranges may lead to problems when plotting or processing such data. Therefore, one of the design criteria of HypPy was to be able to sort the wavelengths of a hyperspectral data cube on-the-fly. In a trade-off between memory use and CPU use, HypPy does not convert the input image into memory; therefore, it must convert the image on-the-fly. Many HypPy tools offer the 'sort_wavelengths' option when opening an image.

HypPy sorts the wavelengths by keeping an index-to-index table as an attribute of the images objects. This table translates the band numbers of an input image into bands that are sorted by wavelength. The advantage is that, internally, the data look like they have their bands sorted by wavelength. The obvious disadvantage is that the band numbers of the input image may not correspond to the band numbers of the original image object.

However, if bands are addressed by wavelengths rather than by band numbers, then this last point is irrelevant. Wavelengths have physical meaning and should be the preferred way of addressing image bands. Band numbers may change and even have a different meaning from one remote sensor to another. Unfortunately, not all images have wavelengths, either because the wavelengths are missing or the images do not contain spectral data, which is why band numbers can still be used in HypPy.

2.3. Viewers

2.3.1. Image Viewer

The image viewer of HypPy offers a no-frills display for viewing hyperspectral images and retrieving spectra. The image viewer was designed for a quick interactive assessment of image quality and the retrieval of endmember spectra. Its simplicity makes it easy to use for novice users of hyperspectral data. Sliders can be used to change the display from one band to another. In color mode, three sliders are available, one for each of the RGB channels of the display. A status bar at the bottom of the window presents information on wavelengths of bands, cursor position, and more. Using the sliders and the left mouse button, the bands and the spectra of the input image can be assessed on quality and information content. Figure 2 shows the hyperspectral image of a rock sample. This particular rock sample 15-91-FE is part of the Fen Complex in Norway, which comprises a carbonatite intrusion [16].

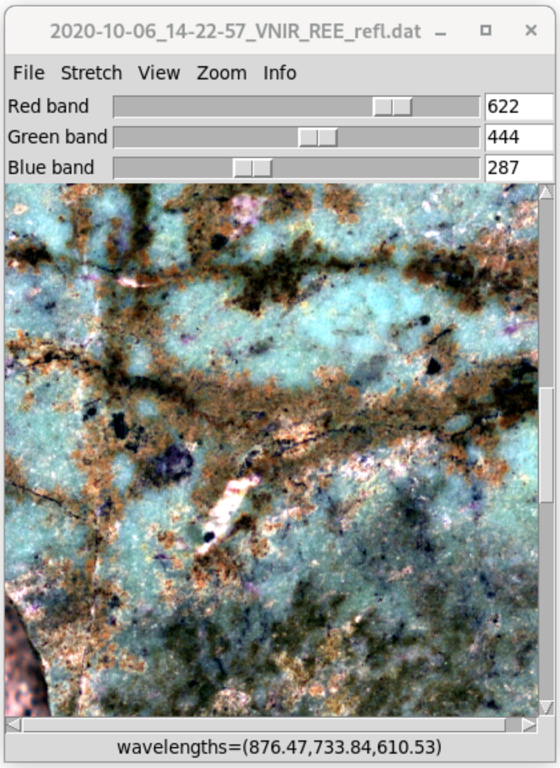

2020-10-06_14-22-57_VNIR_REE_refl.dat
File
Stretch
View
Zoom
Info
Red band
622
Green band
444
Blue band
287
wavelengths=(876.47,733.84,610.53)

Figure 2. Image viewer showing a false color composite of sample 15-91-FE. Red channel 876 nm, green 734 nm, and blue 611 nm. The sliders for the red, green, and blue can be used to change the bands for the RGB color channels.

Under the file menu, two options can be set to control whether the bands must be sorted prior to reading or to use the BBL of an image.

The display program can open two images at the same time. The first image is always visible in the viewer. The second image, called the 'shadow image', is never visible. However, when the shadow image is present, spectra are obtained from the shadow image rather than the visible image. This is like a GIS in which the first layer is on top, and the second layer is hidden below the first image. This option is useful for loading a classification image or mask image in the first layer while spectra are taken from the shadow image, making is possible to assess whether a calculated classification or mask image makes sense. Furthermore, this option is useful for exploring a wavelength map together with a hyperspectral image from which the wavelength map was derived.

The stretch menu offers a number of standard methods for stretching the image [17]. Methods like 1–99% and histogram equalization are implemented. On top of that, the image can be inverted, saturation-enhanced on-the-fly, or pseudo-colored with a number of available color schemes.

A left-click anywhere on the image will bring up a plot window with a graph representing the spectrum of the pixel at this location, see Figure 3. Further clicking on the image will bring up more graphs in the plot. By removing the plot window, the process starts all over with a clear plot window. In this way, the spectra of different points in the image can be visualized together.

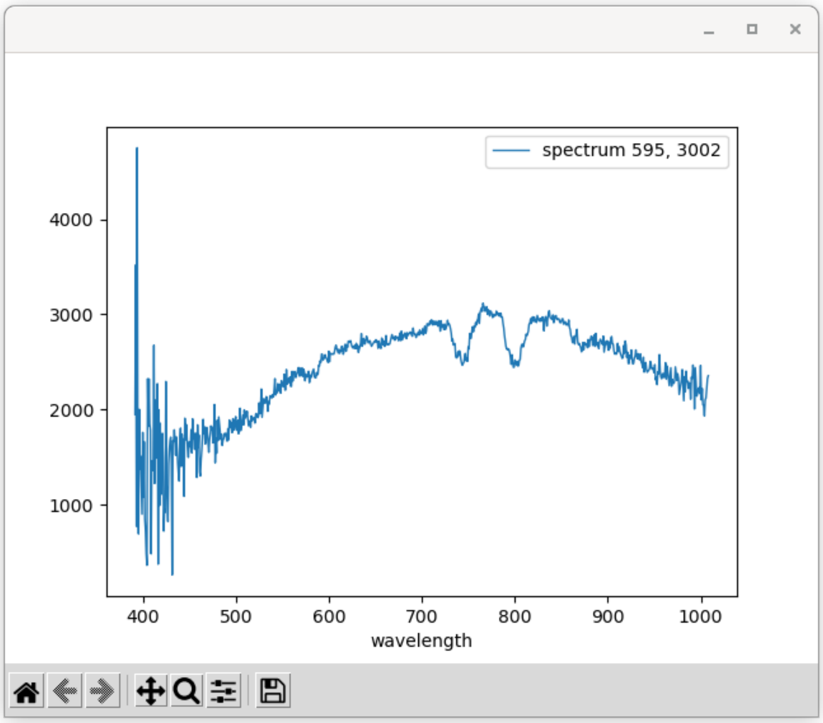


Figure 3. Plot window showing the spectrum of pixel location 595, 3002. The spectrum shows two absorption features at 740 and 800 nm. These features are due to rare earth elements (in particular neodymium) contained by monazite crystals. The spectrum below 450 nm is noisy due to the low sensitivity in the blue of the hyperspectral VNIR camera. The toolbar at the bottom of the window is a standard matplotlib figure toolbar and has buttons for navigating the figure: home, back, forward, pan/zoom, zoom-to-rectangle, and subplot configuration. The save button on the right can be used to save the figure to various formats such as `png` or `pdf`.

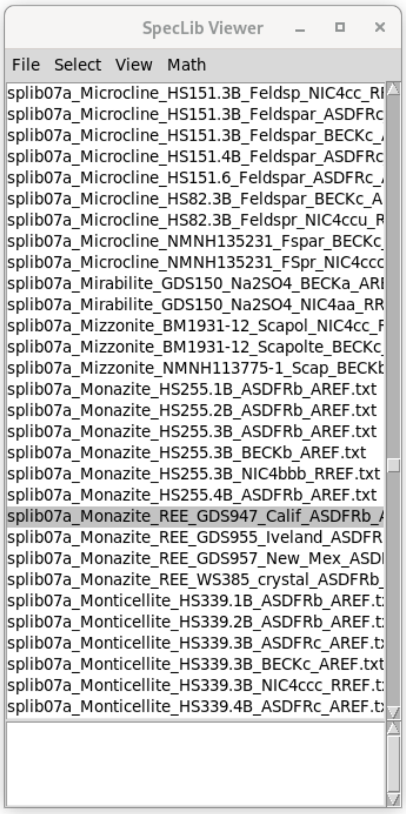
Using the 'Values Window', these spectra can also be saved as a text file, endmembers, or spectral libraries, or imported into other software for visualization and further processing. A directory of endmember spectra can be used as a spectral library in HypPy's classifiers, spectral angle mapper, and linear unmixing.

The HypPy image display program uses the Image and ImageTk modules from the Python Imaging Library (PIL, now called Pillow [18]) for displaying images and matplotlib for making 2D graphs of spectra.

2.3.2. Spectral Library Viewer

The spectral library viewer is a viewer for ENVI-type and ASCII-type spectral libraries. After selecting the input spectral library, the names of the spectra contained by the spectral library are shown in a list. By double-clicking on a name, a plot window will pop up showing the spectrum of the selected mineral or material. Selecting multiple spectra will plot these spectra together in the plot window, enabling a comparison between spectra.

The spectral library viewer supports various ASCII formats, the CRISM spectral library format (PDS), USGS SpecLib version 6 and 7 [12], the ASD format [19], and plain-text two-column data. The spectral library viewer can recursively read an entire directory tree. See Figures 4 and 5, which show the interface of the spectral library viewer and a plot window showing an example spectrum of monazite.



```
splib07a_Microcline_HS151.3B_Feldsp_NIC4cc_RI
splib07a_Microcline_HS151.3B_Feldspar_ASDFRc
splib07a_Microcline_HS151.3B_Feldspar_BECKc_
splib07a_Microcline_HS151.4B_Feldspar_ASDFRc
splib07a_Microcline_HS151.6_Feldspar_ASDFRc_
splib07a_Microcline_HS82.3B_Feldspar_BECKc_A
splib07a_Microcline_HS82.3B_Feldspr_NIC4ccu_R
splib07a_Microcline_NMNH135231_Fspar_BECKc_
splib07a_Microcline_NMNH135231_FSpr_NIC4ccc
splib07a_Mirabilite_GDS150_Na2SO4_BECKa_ARI
splib07a_Mirabilite_GDS150_Na2SO4_NIC4aa_RR
splib07a_Mizzonite_BM1931-12_Scapol_NIC4cc_F
splib07a_Mizzonite_BM1931-12_Scapolte_BECKc_
splib07a_Mizzonite_NMNH113775-1_Scap_BECKt
splib07a_Monazite_HS255.1B_ASDFRb_AREF.txt
splib07a_Monazite_HS255.2B_ASDFRb_AREF.txt
splib07a_Monazite_HS255.3B_ASDFRb_AREF.txt
splib07a_Monazite_HS255.3B_BECKb_AREF.txt
splib07a_Monazite_HS255.3B_NIC4bbb_RREF.txt
splib07a_Monazite_HS255.4B_ASDFRb_AREF.txt
splib07a_Monazite_REE_GDS947_Calif_ASDFRb_
splib07a_Monazite_REE_GDS955_Iveland_ASDFR
splib07a_Monazite_REE_GDS957_New_Mex_ASDI
splib07a_Monazite_REE_WS385_crystal_ASDFRb_
splib07a_Monticellite_HS339.1B_ASDFRb_AREF.t
splib07a_Monticellite_HS339.2B_ASDFRb_AREF.t
splib07a_Monticellite_HS339.3B_ASDFRc_AREF.t
splib07a_Monticellite_HS339.3B_BECKc_AREF.txt
splib07a_Monticellite_HS339.3B_NIC4ccc_RREF.t
splib07a_Monticellite_HS339.4B_ASDFRc_AREF.t
```

Figure 4. Interface of the spectral library viewer, in this case, loaded with the spectral library version 7 from the USGS [12].

Furthermore, the spectral library viewer supports the manipulation of spectra using a 'spectral math' expression. A subset of numpy functions can be applied to spectra. Indices and slices can be used on spectra, and a host of mathematical operators and functions are available for building processing operations on spectra and testing these on (a selection of) a spectral library.

Spectra with different wavelengths and wavelength ranges will automatically be resampled on-the-fly. All these functions are available in the API and are implemented in the spectrum class of HypPy(see also Section 2.4.2 below). Results of the spectral math operation in the spectral library viewer can be saved by setting an output directory first.

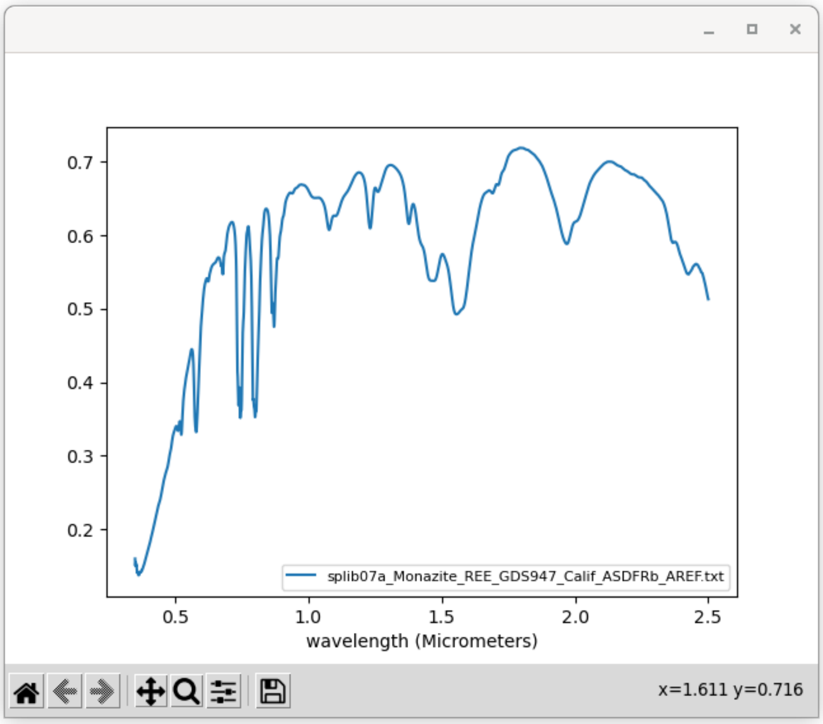


Figure 5. Plot window showing the spectrum Monazite_REE_GDS947_Calif_ASDFRb_AREF. This sample is known to contain Ce, La, Nd, and Th. Notice the Nd absorption features at 740 nm and 800 nm.

2.4. Tools

In the following, a subset of HypPy's tools will be discussed in more detail.

2.4.1. Band Math Tool

Multispectral and hyperspectral images can be regarded as a set of spectral bands, and functions are often created to process such data in a band-by-band fashion. The band math tool implements this.

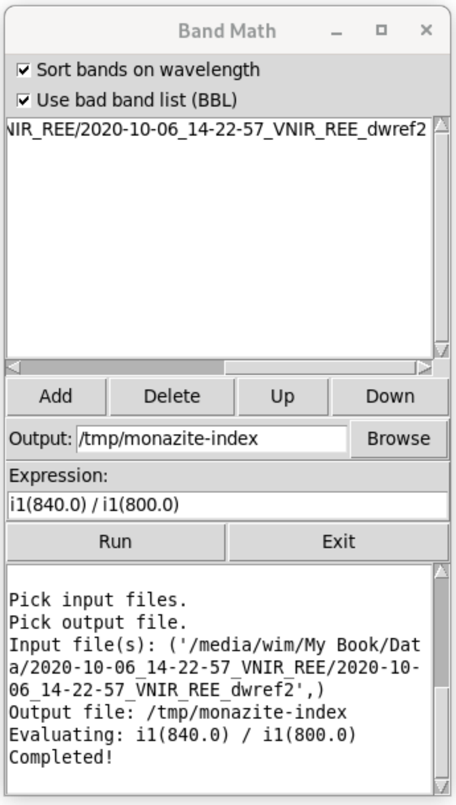
An example of such a band math operation is the calculation of the NDVI, in which the difference between a near-infrared band and a red band is normalized by dividing by the sum of the near-infrared band and the red band. For the NDVI, usually for the red band, a range of 400 nm to 700 nm is used, and for infrared, the range is 700 nm to 1100 nm. In the following example, for red, we pick a band around 675 nm, and for infrared, a band around 1000 nm is chosen. In HypPy, the NDVI of an AVIRISNG image can be calculated using one of the following expressions: (1) or (2).

$$(i1[59] - i1[124])/(i1[59] + i1[124]), \tag{1}$$

$$(i1(675.0) - i1(1000.0))/(i1(675.0) + i1(1000.0)). \tag{2}$$

In band math, the sequence of image file names is linked to variable $i1$, $i2$ and so on. In the example of the NDVI, only one input image is needed, which is linked to the variable $i1$. Images are numbered starting at 1, and band numbers are numbered starting at 0. In the example, band 59 is the band with a center wavelength of 676.9 nm and band 124 has a center wavelength of 1002.4 nm. Please note that, in this example, the BBL is used, and the first seven bands, which are marked as bad, are not present.

In band math, it is also possible, and even preferred, to use wavelengths instead of band numbers. The band with the nearest central wavelength will be automatically selected. Note that, in band math, when wavelengths are used rather than band numbers, these wavelengths must be enclosed by round brackets, not square brackets. In Python, this is implemented using a 'call' on the image class. Python objects can be designed to behave like lists or iterators, having an index between square brackets or a function with a parameter list between round brackets. Figure 6 shows an example of a band ratio.



```
Pick input files.
Pick output file.
Input file(s): ('/media/wim/My Book/Dat
a/2020-10-06_14-22-57_VNIR_REE/2020-10-
06_14-22-57_VNIR_REE_dwref2',)
Output file: /tmp/monazite-index
Evaluating: i1(840.0) / i1(800.0)
Completed!
```

Figure 6. Interface of the band math tool. The formula shows an example of a band ratio for a simple monazite index based on the shoulder and the absorption feature of Nd at 840 nm and 800 nm, respectively. In this case, there is only one input file *i*1.

HypPy uses the Python eval() function to execute expressions like the NDVI example above. Band math expressions are basically Python expressions. HypPy sets up the variables *i*1, *i*2, and so on. These are HypPy image objects, which can be called to return bands of a certain wavelength. Linking the image variables to image files is an essential step in the use of band math expressions. Note that the syntax used in band math differs from the one used in spectral math. Band math uses image objects, while spectral math uses spectrum objects; see below.

2.4.2. Spectrum Class and Spectral Math

Spectral math is used to apply mathematical expressions to spectra of images or spectral libraries. Band math uses the band paradigm; spectral math uses the spectrum paradigm. In spectral math, the image is processed pixel by pixel. In band math, the image is processed band by band. Spectral math relies on the functionality as it is implemented in the spectrum class. The interface of the spectral math tool is similar to the band math tool (see Figure 6), but the expression must follow the syntax of the spectrum class.

The spectrum class was created for the manipulation of spectra. The idea is that a spectrum object contains information about wavelengths as well as values, be it reflectance, radiance, or other data. The spectrum class offers access to the data, as well as functions to operate on the spectral data.

The list below summarizes some of the properties of the spectrum container object:

- The spectrum object contains wavelengths and values.
- The spectrum object may contain a name and a description.

- An index can be used to select values or make subsets of the spectrum.
- An integer as index denotes a band number; a float denotes a wavelength.
- Mathematical operators like +, −, * and/can be used on spectra.
- The spectrum object contains a wealth of operators and methods. Currently, some 150 functions and 20 operators are implemented (see HypPy's spectral math manual).
- When working with spectra with different wavelengths, the spectra are automatically resampled on-the-fly.

As shown previously, spectral indices can be implemented using band math. A spectral index is supposed to have a high value if a certain mineral or material is present and a low value if it is absent. An example of such an index was given in the previous section in which we presented the example of the NDVI.

However, if the image contains noise, then the resulting NDVI may also contain noise. In such a case, it may be beneficial to work with average values in band ranges rather than single bands. In band math, it is not possible to use band ranges because bands of a range must be addressed one by one. However, in spectral math, a spectral range can be obtained using a slice on the spectrum.

The spectral math expression for NDVI may now look like the following expression (3):

$$\begin{aligned}&(S1[700.0:1100.0].mean() - S1[400.0:700.0].mean())/\\&(S1[700.0:1100.0].mean() + S1[400.0:700.0].mean()),\end{aligned} \tag{3}$$

in which $S1$ denotes the spectra from the first image. Spectra of subsequent input images are numbered $S2$, $S3$, and so on. However, in this example, only one image is used. The square brackets contain what in Python are called slices, denoted by the use of a colon ':'. Notice that floats are used as indices, which denote wavelengths. Integers as indices are interpreted as band numbers. The result of a slice on a spectrum is a new spectrum object that contains a subset of the original spectrum object. In this case, the subset will contain the spectral values roughly between 400 nm and 700 nm and between 700 nm and 1100 nm.

The wavelengths are approximate because these wavelengths are matched to the closest band numbers, which may not have the exact same wavelengths as central wavelengths. HypPy does not generate errors or warnings for the conversion of wavelengths to bands to prioritize speed over quality checks, as the checks would consume significant CPU time. However, users and programmers can verify the requested and obtained wavelengths using the functions wavelength2index() and index2wavelength().

Subsequently, the averaging function mean() is used on these subsets. In other words, the result will be the NDVI calculated from the average spectrum between 400 nm and 700 nm and the spectrum between 700 nm and 1100 nm. In this way, spectral indices that operate on ranges of wavelengths rather than single bands can be built. The obvious advantage is that this reduces noise in the output. The downside is that such expressions are slower than band-by-band operations. Furthermore, in the example above, the slices and means are calculated twice because the spectral math tool only supports one single expression. However, by making clever use of the walrus operator ':=', which was introduced in Python version 3.8 [20], double calculation can be prevented. The expression for the NDVI would then look like expression (4):

$$\begin{aligned}&((ir := S1[700.0:1100.0].mean()) - (red := S1[400.0:700.0].mean()))/\\&(ir + red),\end{aligned} \tag{4}$$

in which the mean of the infrared range is stored in the variable *ir* and the mean of the red range is stored in the variable *red*, respectively, and these values are retrieved later in the expression for calculating the denominator.

Please note that we use a broad range for red and infrared in Equations (3) and (4), using the chlorophyll absorption in the visible part of the spectrum (400 nm to 700 nm) as *red* and the reflection by the leaf cell structure in the near-infrared part of the spectrum

(700 nm to 1100 nm) as *ir*. The specific wavelength ranges may vary depending on the remote sensing sensor or technology in use. For instance, Tucker [21] used the range 630 nm to 690 nm for *red* and 750 nm to 800 nm for *ir*, which are the ranges of the Landsat MSS spectral bands 5 and 7, respectively.

HypPy uses the Python eval() function to execute expressions like the averaging NDVI example above. The spectral math expressions are Python expressions. However, HypPy sets up the variables $S1$, $S2$, and so forth, which are HypPy spectrum objects which can be used as a Python container class. In that sense, the behavior of the spectrum object is similar to a Python list object, which can be indexed, sliced, looped over, and so on.

In HypPy, spectral math can be used in the spectral math tool as well as in the spectral library viewer.

2.4.3. Spatio-Spectral Filters

Spatio-spectral filters take into account both spectral information and spatial information. HypPy has three types of spatio-spectral filters: gradient filters, mean filters, and binning filters. Gradient filters are implemented by regarding a local spatial kernel of spectra as vectors and to calculate the weighted difference between these vectors. The mean filters calculate the weighted sum of these vectors. Binning filters reduce the number of samples by aggregating values.

HypPy has a tool for hyperspectral edge filtering. Such filters can be used to detect spectral variability in a scene. Areas with a high value are spectrally inhomogeneous, and areas with low values are spectrally homogeneous. The results of such filters can serve as a first step in image quality assessment, classification, or segmentation. The result of the gradient filters is always a single-band grey-level image. The HypPy tool called hyperspectral gradient implements a number of directional (up, down, and two diagonals) and non-directional filters (edgy and Sobel). A number of distance measures have been implemented that can be used in conjunction with these filters.

The spectral angle (SA) distance measure only takes into account the spectral information of the input image because the spectral angle is insensitive to the intensity of the input spectrum. The intensity difference (ID) distance measure, on the other hand, only takes into account the intensity information of the spectrum, not the spectral information. The Euclidean distance (ED) measure is sensitive to both the spectral and intensity information of the spectra, although most of the time, the ED seems to be dominated by the intensity of the input spectra. In addition to the standard spectral difference measures discussed above, two other distance measures, which are widely used in hyperspectral analysis, are available: the Bray–Curtis (BC) distance and the spectral information divergence (SID) [22]. All these distance measures have their own properties regarding spectral similarity.

Where the gradient filters take into account a 3×3 two-dimensional set of full spectra, the mean filters take into account a smaller three-dimensional set of spectral values. The mean filters apply a $3 \times 3 \times 3$ kernel on the hyperspectral data cube. The filter can take all the values of the kernel into account or a limited set of the kernels. Current kernels in HypPy calculate the weighted average of 27, 19, or 7 of the values of the kernel. Two averaging functions are implemented: the mean and median. A mean filter is a linear filter and, therefore, predictable in what the effect on the image will be. A median filter reduces the effect of outliers. In that respect, a median filter may be useful for filtering out pepper-and-salt noise. However, the effect of a median filter is not as easy to predict as a mean filter, which may lead to unexpected effects in the output image. In certain cases, median filters may have an effect comparable to morphological filters in the sense that areas seem to shrink or grow, which may be confusing and thus complicate the interpretation of the resulting image.

The result of a spatio-spectral mean filter is always a hyperspectral cube with the same dimensions as the input. However, like with regular two-dimensional spatial filters, edge effects should be taken into account. The first and last values in any of the three dimensions (x, y, and *band*) are treated differently than the rest of the hyperspectral cube.

In addition to these filters, HypPy has a number of binning tools, spatial, spectral, and spatial–spectral binning, to reduce the number of samples in the image cube. These tools can be used to decrease the data volume, reduce the noise by averaging, and speed up subsequent processing steps. For instance, spatial–spectral binning of $xy = 3 \times 3$ and $z = 11$ will reduce the data volume by almost a factor of 100 ($3 \times 3 \times 11 = 99$) and reduce the standard deviation of the noise by almost a factor of 10 ($\sqrt{99}$).

2.4.4. Wavelength Mapper

Mapping physical properties from hyperspectral images is complicated and often relies on the availability of spectral libraries of target materials. Many existing methods depend on the subjective decisions of the interpreter for the selection of endmembers and the post-processing of rule images. If no prior information is available, information should be extracted by characterization of spectral absorption features. On multi-spectral images, such features can be found using indices based on band ratios and band depths; see, for instance, [23]. However, hyperspectral images show more spectral detail of the features, which opens up the possibility of using more sophisticated methods.

Absorption features of spectral curves are important for analysis and interpretation. Such features may be parametrized as center wavelength position, depth, area, and asymmetry [24]. Usually, the most important features are the center wavelength position and the depth of the main absorption feature. The wavelength position of an absorption feature can be used for the identification of minerals because many minerals have their own unique absorption wavelengths. The depth of such a feature is an indication of the abundance of a mineral.

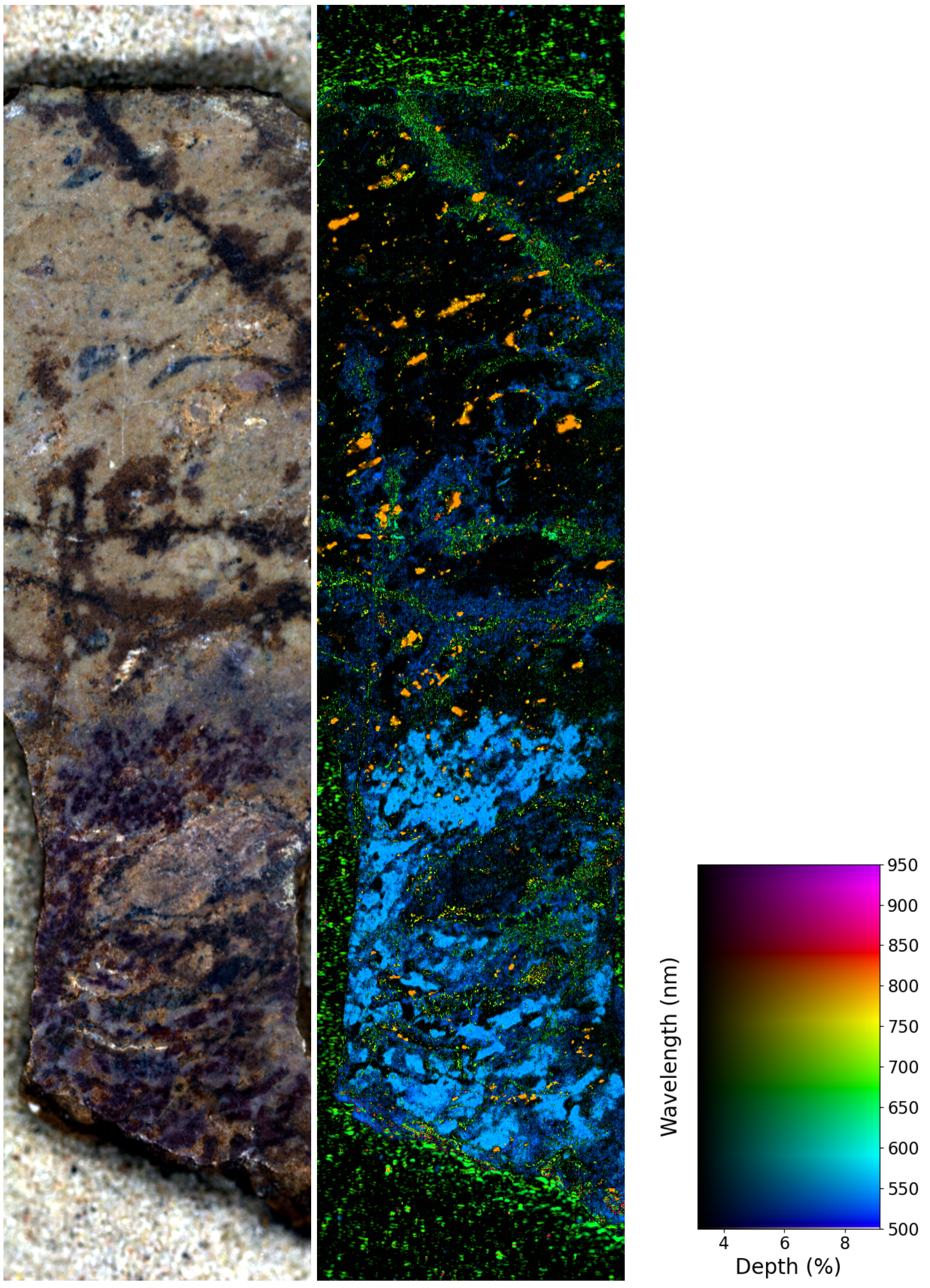
The center wavelength and depth of the deepest feature and consecutive deepest features can automatically be determined. These parameters can be calculated for all the pixels (spectra) of a hyperspectral image and presented as grey-level images representing the wavelength and depth of the feature.

Subsequently, the resulting wavelength and depth images can be fused using an HSI (hue–saturation–intensity) transform [17] in such a way that the resulting image shows the wavelength in color (the hue). The intensity represents the depth of the feature. The saturation is set to 1. The wavelength mapper tool of HypPy automatically determines the wavelength and depth of the deepest absorption feature of all the pixels of a hyperspectral cube and fuses the two into one color image.

Figure 7a shows a rock sample in a natural color composite. Figure 7b shows the result of the wavelength mapper; Figure 7c shows the legend to the wavelength map. The REE-containing monazite is visible as orange grains. The greens and the blues are carbonates. For a further discussion of this rock sample, see [16].

The wavelength mapper is unsupervised, straightforward, and repeatable; large areas can be mapped at once, and details are preserved within the focus of the chosen spectral range. The result, called the wavelength map, can be understood by users with a limited image processing background. However, although the colors may suggest the presence of certain minerals, the resulting wavelength map is not a mineral map. The wavelength map can be used as a pre-classification step in order to understand the spectral variability in the scene. The knowledge obtained from the wavelength map is useful for the additional steps of producing a mineral map.

The wavelength mapper relies on only two input parameters: the wavelength range to focus on, and the stretch to use for the depth of the feature. The wavelength mapper was first published in van Ruitenbeek et al. [25], and a tutorial on the method was published in [24].



(**a**) Color Composite (**b**) Wavelength Map (**c**) Legend

Figure 7. (**a**) Natural color composite of rock sample 15-91-FE (red 650 nm, green 550 nm, and blue 450 nm). (**b**) The wavelength map of the range of 500 nm to 950 nm shows the wavelength and depth of the deepest feature. Monazite is visible in orange (showing the 800 nm feature of Nd) or yellow–green when the 740 nm feature happens to be deeper than the 800 nm feature. (**c**) Legend to the wavelength map.

2.4.5. Decision Trees

A decision tree is a type of classifier that consists of a series of binary decisions used to determine the class for each input pixel. The decisions are based on the characteristics of the datasets. Every decision partitions data into either one of two potential classes or groups of classes. Decisions are expressions applied to a single image or a set of images. Each of these images represents a property of the input pixels.

One such property, for instance, could be the wavelength position of the deepest absorption feature of the spectrum of the input pixel. An expression could be built to check whether the input pixel might contain a clay mineral with a deepest absorption feature of around 2.2 µm. Subsequent decisions could check the water features of this pixel to see whether the material has a high crystallinity index. After a number of decisions, a class will be assigned to the pixel.

Decision trees are useful for building classifiers that are based on physical properties rather than statistical inference. Of course, this can only be carried out if all the properties needed for the decision trees can be established with enough confidence.

A package like ENVI supports the construction and visualization of new decision trees. However, ENVI has a number of disadvantages. The way decision trees are implemented in ENVI is rather rigid because once a node in the tree is constructed, its place in the decision tree is fixed. In a finalized ENVI decision tree, it is impossible to move elements around. In ENVI, if you want to change the decision tree, you have to rebuild it from scratch.

HypPy can be used to execute ENVI decision trees. The only condition is that expressions used for decisions can be translated into expressions in Python.

In addition to the ENVI decision tree format, HypPy has its own decision tree format. This format is a text-based format that can be edited using a text editor.

For the visualization of decision trees, HypPy uses the Graphviz package [26]. To this end, the ENVI or HypPy decision tree must be converted to the 'dot' format first. The dot language is an extensive language for describing graphs. The binary decision trees used by ENVI and HypPy present a tiny subset of all possible graphs and types of graphs supported by the Graphviz package.

2.4.6. Zonal Statistics

The zonal statistics tool calculates statistics on pixels of an input image within the zones defined by the zone image. The input image is a regular hyperspectral or multispectral image. The zone image can be the result of a classification image or a mask image. For each distinct value in the zone image, an aggregation function is run on the input image for all the locations having this distinct value in the zone image. The currently implemented aggregation functions are as follows: mean, median, minimum, maximum, standard deviation, and mean plus or minus two times the standard deviation.

The results of the zonal statistics are useful for determining the properties of areas in the input image. For instance, in the case of noise in the input image, the mean spectrum of an area may reveal more spectral detail than the spectrum of a single pixel, in which spectral details of features may be drowned out by noise.

3. Application Areas

Below, the application of HypPy in several projects in education, consulting, and research will be discussed in more detail.

3.1. Education

HypPy is one of the software packages used in our spectral geology courses at the University of Twente [27]. The clearness of the HypPy interface makes it an excellent tool for students starting out in spectral geology. One example of published student research using HypPy is [28]. For a tutorial on analyzing absorption features for finding ore using geological remote sensing, see [24].

HypPy can be used in the short courses to learn the basic concepts of hyperspectral imaging and carry out specialized operations on hyperspectral data processing and interpretation. For example, HypPy was used in the course 'Remote Sensing and Mineral Spectroscopy for the Exploration and Mining Geologist' offered in Istanbul [29].

In December 2022, HypPy was used in a spectral workshop during the Annual GRSG Conference [30]. The workshop comprised an overview of the theory and practical implementation of spectral methods for mineral identification, accompanied by a hands-on laboratory session involving the application of spectral processing analytics to a recently obtained high-resolution airborne hyperspectral dataset that spans the visible and near-infrared spectra (VNIR 400 nm to 1 µm), short-wave infrared (SWIR 1 µm to 2.5 µm), and long-wave infrared (LWIR 8 µm to 15 µm) spectral ranges. The laboratory session employed the HypPy hyperspectral processing software and provided unhindered access to sample data.

3.2. Consulting

We explored the viability of employing laboratory-based short-wave infrared (SWIR) hyperspectral imagery for a commercial partner to identify and map the mineralogical composition of specific sedimentary drill core samples from the subsurface of the Netherlands. The scripting capabilities of HypPy proved essential in this project.

Our findings indicate that utilizing laboratory-based short-wave infrared (SWIR) hyperspectral images with a spatial sampling of 200 µm offers a valuable non-destructive approach for identifying and mapping white mica, clay, and carbonate minerals in sedimentary drill cores. Together with partners at the University of Groningen, using HypPy we succeeded in building a prototype processing chain for the quantification of the mineral content and deriving a number of geophysical properties of the drill cores.

3.3. Research

The first work on HypPy started with the Mars Planetary Mapping Pilot Project for the European Space Agency (ESA). The pilot study was designed to transfer the modern, digital methods and workflows used in terrestrial geological mapping to planetary mapping. The focus of this study was placed on the ESA datasets from the High Resolution Stereo Camera (HRSC) and the Visible and Infrared Mineralogical Mapping Spectrometer (OMEGA) hyperspectral instrument onboard the Mars Express [31]. The first step of the workflow was to process the raw data to highlight features of interest in derived information products. These included digital elevation models extracted from the stereo data and mineral maps made using the hyperspectral data. These tools were used to interpret the data and produce a geological map for the Nili Fossae area. This study paved the way for the wavelength mapping technique to be adopted for mapping the planets Mars and Earth on a routine basis [7].

HypPy introduces a processing methodology designed to transform OMEGA data from radiance-at-sensor to surface reflectance. To address particular challenges associated with OMEGA data, we devised innovative approaches for determining a per-scene transmittance spectrum, conducting atmospheric correction, and filtering systematic and random noise. See, for instance, van Ruitenbeek et al. [25]. These filters are now part of HypPy. The processing methodology generates surface reflectance images, allowing the extraction of single-pixel spectra for surface feature identification. In alignment with various prior studies, our findings suggest, among other observations, the existence of iron-rich clays on the Martian surface.

Mapping physical properties from hyperspectral images is complicated and frequently depends on the accessibility of spectral libraries of the target materials. Many existing methods depend on the subjective decisions of the interpreter for the selection of endmembers and the postprocessing of rule images. If no prior information is available, information should be extracted by the characterization of spectral absorption features. In van Ruitenbeek et al. [25], they describe a method called the wavelength mapper (see Section 2.4.4) to automatically determine the minimum reflectance within a wavelength range. We conclude that the wavelength mapper proves to be a useful tool for unbiased endmember selection, mineral mapping, and the interpretation of areas demonstrating zonation of mineral alterations using hyperspectral imagery. The usefulness of the wavelength mapper has already been recognized in other research. For instance, it was used to map mineral outcrops of lithium-bearing pegmatites in a pilot quarry located near Uis, Namibia [32].

In another study, the entropy function was added to the spectrum class to enable the calculation of an index for sorting processes in hydrothermal systems. Hydrothermal processes affect the composition of rocks by chemical reactions and fluid flow, and this paper showed that the remaining patterns of such processes can be visualized using the Shannon entropy. The application of entropy is useful for studying hydrothermal mineral deposits and may be valuable in studies of early life environments as well.

Functions that have already been implemented in the spectrum class of HypPy could be used to add noise to data. This was used for the simulation of airborne SWIR hyperspectral sensors for the mapping of methane (CH_4) gas [33]. The study's findings indicated that three presently active airborne imaging spectrometers were unsuccessful in detecting the CH_4 plume. The paper emphasized the importance of both spectral sampling and radiometric calibration for the successful mapping of methane.

Contemporary imaging spectrometers are compact, lightweight, cost-effective, and applicable in laboratory settings. Precise hyperspectral images necessitate minimal noise, precise calibration, and consistent response uniformity. Periodic checks are advisable due to the aging of optical and electronic components. In another study we introduce a technique for quantifying smile and keystone geometric distortions from test images [34]. The measurements are acquired through the utilization of straightforward models for smile and keystone, along with pre-existing tools in HypPy. Additionally, functions for quadratic spline interpolation, normalized cross-correlation, and optimization are applied to obtain the smile and keystone values.

Findings indicate that the smile and keystone of a laboratory imaging spectrometer can be determined with sub-pixel precision.

4. Discussion

In addition to HypPy, several other packages offer functionality for processing hyperspectral images.

To name a few, QUANTools was implemented using the ENVI/IDL programming language for the automatic detection of multiple absorption feature parameters [35]. However, to run QUANTools, a runtime version of ENVI/IDL is needed, and the QUANTools package offers limited functionality, similar to one specific tool in HypPy, which is called the wavelength mapper.

PySptools is a Python module that implements spectral and hyperspectral algorithms [36]. However, this is a package or library, and you have to do coding before you can use it. In fact, HypPy uses pysptools for its linear unmixing classifier. Unlike pysptools, HypPy offers a user interface to the functionality.

An additional noteworthy toolkit is known as spectral Python (SPy) [37]. SPy is a Python module designed for handling hyperspectral image data, featuring functions for tasks such as reading, displaying, manipulating, and classifying hyperspectral imagery. Notably, SPy is limited to interactive usage from the Python command prompt or through Python scripts, unlike HypPy.

The Orfeo and EnMAPBox software packages are both powerful tools used in remote sensing and image processing applications [38,39]. They are widely used in the remote sensing community and provide extensive capabilities for processing and analyzing satellite imagery. While these packages focus on the processing of satellite images, HypPy is also targeted at processing general hyperspectral images, such as proximal images recorded with hyperspectral cameras in the laboratory.

Just like the Python language itself [6], HypPy was designed with simplicity rather than performance in mind [40]. Some operations in HypPy may be relatively slow. However, the modularity of the functions and tools can be used to build experimental programs or even prototypes of processing chains that run on any hardware.

Regarding speed, future versions of Python may be faster. To name one ongoing project, the Mojo language is a superset of the Python language with the speed of the C language [41].

5. Conclusions

In this paper, we described the design, implementation, and application of the Python package called Hyperspectral Python, HypPy. The idea of HypPy is to be able to process hyperspectral images using free and open-source software. HypPy was developed using the Python language and relies heavily on the array-processing capabilities of packages like NumPy and SciPy. HypPy can be used for early data analysis and prototyping of processing chains. HypPy was designed with simplicity and practical imaging spectrometry in mind. Simultaneously, it incorporates several innovative and potent concepts.

HypPy is a package in development and for development, and has proven its merits. Its simplicity and flexible top-level menu structure makes HypPy attractive for use in education and short courses. New methods for research can quickly be programmed and tested. Examples of such research projects are the ESA Mars planetary mapping pilot project and the research project for determining the keystone and smile of hyperspectral line cameras.

In projects, it has been used to build entire processing chains from hyperspectral data to information. In short, HypPy can be used as a platform for generating and testing new ideas for the processing of hyperspectral data. The core of reading and writing hyperspectral images, the wavelength mapper, hyperspectral gradient filters, spatio-spectral filters, band math and spectral math tools with their own expression language, decision tree tools, the use of zonal statistics, and dedicated image and spectral library viewers are good examples of that.

We expect HypPy to also be of great added value to other organizations and individuals in research, education, and projects using hyperspectral data.

Author Contributions: Conceptualization, W.B. and F.v.R.; methodology, W.B.; software, W.B.; validation, W.B., F.v.R., H.v.d.W. and C.H.; resources, A.D.; writing—original draft preparation, W.B.; writing—review and editing, H.v.d.W.; visualization, W.B.; supervision, H.v.d.W. and F.v.d.M.; project administration, W.B.; funding acquisition, F.v.d.M. and F.v.R. All authors have read and agreed to the published version of the manuscript.

Funding: We acknowledge the financial support received from the European Space Agency (the Mars planetary mapping project) and Wijnand van Eindthoven of Deep Atlas. The funding provided by these organizations played a crucial role in the successful execution of our research project.

Data Availability Statement: HypPy can be downloaded from Github and Zenodo, see [42]; Name: HypPy (Hyperspectral Python); License: GPU General Public License Version 3; Contact: w.h.bakker@utwente.nl; Hardware requirements: regular windows, mac, or linux laptop or desktop; Program language: Python version 3; Minimum software requirement: numpy, scipy, pillow, matplotlib, pysptools; Program size: 12.2 MB.

Acknowledgments: We are grateful to the University of Twente for furnishing the essential resources and facilities that greatly supported the seamless advancement of our research. Furthermore, we are grateful to all students who directly or indirectly contributed to this work by using HypPy during their studies. Their collaboration and encouragement were instrumental in the shaping of HypPy.

Conflicts of Interest: The authors declare no conflicts of interest.

Abbreviations

The following abbreviations are used in this manuscript:

API	A Programming Interface
BBL	Bad Band List
BIL	Band Interleaved by Line
BSQ	Band Sequential
CLI	Command-Line Interface
GUI	Graphical User-Interface
SWIR	Shortwave Infrared
VNIR	Visible and Near-Infrared

References

1. NV5 Geospatial Solutions Inc. ENVI—Environment for Visualizing Images. 2023. Available online: https://www.nv5geospatialsoftware.com/docs/using_envi_Home.html (accessed on 24 November 2023).
2. Prokakis, E. Free and Open-Source Software: Freedom, Transparency and Efficiency in the Digitalization Era. *J. Politics Ethics New Technol. AI* **2022**, *1*, e31230. [CrossRef]
3. Harris, C.R.; Millman, K.J.; van der Walt, S.J.; Gommers, R.; Virtanen, P.; Cournapeau, D.; Wieser, E.; Taylor, J.; Berg, S.; Smith, N.J.; et al. Array programming with NumPy. *Nature* **2020**, *585*, 357–362. [CrossRef] [PubMed]
4. Virtanen, P.; Gommers, R.; Oliphant, T.E.; Haberland, M.; Reddy, T.; Cournapeau, D.; Burovski, E.; Peterson, P.; Weckesser, W.; Bright, J.; et al. SciPy 1.0: Fundamental algorithms for scientific computing in Python. *Nat. Methods* **2020**, *17*, 261–272. [CrossRef] [PubMed]
5. Hunter, J.D. Matplotlib: A 2D Graphics Environment. *Comput. Sci. Eng.* **2007**, *9*, 90–95. [CrossRef]
6. van Rossum, G.; Drake, F.L. *Python 3 Reference Manual*; CreateSpace: Scotts Valley, CA, USA, 2009. Available online: https://dl.acm.org/doi/book/10.5555/1593511 (accessed on 31 July 2024).
7. Tragheim, D.G.; Marsh, S.H.; Pedley, R.C.; Napier, B.R.; Bateson, L.; Smith, A.G.; Marchant, A.P.; Gunnink, J.L.; Oosthoek, J.H.P.; Muller, J.P. *The Mars Planetary Mapping Pilot Project*; European Space Agency (ESA): Paris, France, 2010.
8. Konstantinides, K.; Rasure, J.R. The Khoros software development environment for image and signal processing. *IEEE Trans. Image Process.* **1994**, *3*, 243–252. [CrossRef] [PubMed]
9. Young, M.; Argiro, D.; Kubica, S. Cantata. *ACM SIGGRAPH Comput. Graph.* **1995**, *29*, 22–24. [CrossRef]
10. Ousterhout, J.K.; Jones, K.; Foster-Johnson, E.; Fellows, D.; Griffin, B.; Welton, D. *Tcl and the Tk Toolkit*, 2nd ed.; Addision-Wesley Professional Computing Series; Addison-Wesley: Upper Saddle River, NJ, USA, 2009.
11. Roseman, M. *Modern Tkinter for Busy Python Developers: Quickly Learn to Create Great Looking User Interfaces for Windows, Mac and Linux Using Python's Standard GUI Toolkit*; Late Afternoon Press: Victoria, BC, Canada, 2020.
12. Kokaly, R.F.; Clark, R.N.; Swayze, G.A.; Livo, K.E.; Hoefen, T.M.; Pearson, N.C.; Wise, R.A.; Benzel, W.M.; Lowers, H.A.; Driscoll, R.L.; et al. *USGS Spectral Library Version 7*; Technical report; USGS: Reston, VA, USA, 2017. [CrossRef]
13. Bibring, J.P.; Langevin, Y.; Altieri, F.; Arvidson, R.; Beilud, G.; Berthél, M.; Douté, S.; Drossart, P.; Encrenaz, T.; Forget, F.; et al. *OMEGA: Observatoire pour la Minéralogie, l'Eau, les Glaces et l'Activité*; European Space Agency, ESA (Special Publication): Paris, France, 2009; SP-1291; pp. 75–95. Available online: https://sci.esa.int/s/WnjXOYW (accessed on 31 July 2024).
14. Vane, G.; Green, R.O.; Chrien, T.G.; Enmark, H.T.; Hansen, E.G.; Porter, W.M. The airborne visible/infrared imaging spectrometer (AVIRIS). *Remote Sens. Environ.* **1993**, *44*, 127–143. [CrossRef]
15. Jia, G.; Hueni, A.; Tao, D.; Geng, R.; Schaepman, M.E.; Zhao, H. Spectral super-resolution reflectance retrieval from remotely sensed imaging spectrometer data. *Opt. Express* **2016**, *24*, 19905–19919. [CrossRef] [PubMed]
16. Marien, C.; Dijkstra, A.H.; Wilkins, C. The hydrothermal alteration of carbonatite in the Fen Complex, Norway: Mineralogy, geochemistry, and implications for rare-earth element resource formation. *Mineral. Mag.* **2018**, *82*, S115–S131. [CrossRef]
17. Richards, J.A.; Jia, X. *Remote Sensing Digital Image Analysis*; Springer: Berlin/Heidelberg, Germany, 1999. [CrossRef]
18. Driscoll, M. *Pillow: Image Processing with Python*; Leanpub: Victoria, BC, Canada, 2021. Available online: https://leanpub.com/pillow (accessed on 31 July 2024).
19. Malvern Panalytical Ltd. ASD File Format—Version 8. Technical Report, Malvern Panalytical. 2017. Available online: https://www.malvernpanalytical.com/en/learn/knowledge-center/user-manuals/asd-file-format-v8 (accessed on 31 July 2024).
20. Hettinger, R. What's New In Python 3.8. 2019. Available online: https://docs.python.org/3/whatsnew/3.8.html (accessed on 24 November 2023).
21. Tucker, C.J. Red and photographic infrared linear combinations for monitoring vegetation. *Remote Sens. Environ.* **1979**, *8*, 127–150. [CrossRef]
22. Chein-I Chang. Spectral information divergence for hyperspectral image analysis. In Proceedings of the IEEE 1999 International Geoscience and Remote Sensing Symposium, IGARSS'99 (Cat. No.99CH36293), Hamburg, Germany, 28 June–2 July 1999; Volume 1, pp. 509–511. [CrossRef]
23. Hewson, R.D.; Cudahy, T.J.; Mizuhiko, S.; Ueda, K.; Mauger, A.J. Seamless geological map generation using ASTER in the Broken Hill-Curnamona province of Australia. *Remote Sens. Environ.* **2005**, *99*, 159–172. [CrossRef]
24. Hecker, C.; van Ruitenbeek, F.J.A.; van der Werff, H.M.A.; Bakker, W.H.; Hewson, R.D.; van der Meer, F.D. Spectral Absorption Feature Analysis for Finding Ore: A Tutorial on Using the Method in Geological Remote Sensing. *IEEE Geosci. Remote Sens. Mag.* **2019**, *7*, 51–71. [CrossRef]
25. van Ruitenbeek, F.J.A.; Bakker, W.H.; van der Werff, H.M.A.; Zegers, T.E.; Oosthoek, J.H.P.; Omer, Z.A.; Marsh, S.H.; van der Meer, F.D. Mapping the wavelength position of deepest absorption features to explore mineral diversity in hyperspectral images. *Planet. Space Sci.* **2014**, *101*, 108–117. [CrossRef]
26. Ellson, J.; Gansner, E.R.; Koutsofios, E.; North, S.C.; Woodhull, G. Graphviz and Dynagraph—Static and Dynamic Graph Drawing Tools. In *Graph Drawing Software*; Springer Link: New York, NY, USA, 2004; pp. 127–148. [CrossRef]
27. ITC. Master of Science Degree Programme in Geo-Information Science and Earth Observation—Academic Year 2023–2024. 2023. Available online: https://studyguide.itc.nl/m-geo (accessed on24 November 2023).

28. Portela, B.; Sepp, M.D.; van Ruitenbeek, F.J.A.; Hecker, C.; Dilles, J.H. Using hyperspectral imagery for identification of pyrophyllite-muscovite intergrowths and alunite in the shallow epithermal environment of the Yerington porphyry copper district. *Ore Geol. Rev.* **2021**, *131*, 104012. [CrossRef]
29. Portela, B. Short Course—Remote Sensing and Mineral Spectroscopy for the Exploration & Mining Geologist November 7–11, 2022, Istanbul. 2022. Available online: https://www.linkedin.com/posts/brunobvportela_geologicalremotesensing-istanbul-turkey-activity-7001884993555693568-bkXG (accessed on24 November 2023).
30. Geological Remote Sensing Group. GRSG Annual Conference & AGM 2022: Orbit to Outcrop. 2022. Available online: https://www.grsg.org.uk/grsg-agm-conference-2022 (accessed on 24 November 2023).
31. Chicarro, A.; Martin, P.D.; Trautner, R. The Mars Express mission: An overview. In *Mars Express: A European Mission to the Red Planet*; ESA: Noordwijk, The Netherlands, 2004; Volume SP-1240. Available online: https://sci.esa.int/documents/33745/35957/1567254632829-OverviewWeb.pdf (accessed on 31 July 2024).
32. Booysen, R.; Lorenz, S.; Thiele, S.T.; Fuchsloch, W.C.; Marais, T.; Nex, P.A.M.; Gloaguen, R. Accurate hyperspectral imaging of mineralised outcrops: An example from lithium-bearing pegmatites at Uis, Namibia. *Remote Sens. Environ.* **2022**, *269*, 112790. [CrossRef]
33. Scafutto, R.D.M.; van der Werff, H.; Bakker, W.H.; van der Meer, F.; de Souza Filho, C.R. An evaluation of airborne SWIR imaging spectrometers for CH4 mapping: Implications of band positioning, spectral sampling and noise. *Int. J. Appl. Earth Obs. Geoinf.* **2021**, *94*, 102233. [CrossRef]
34. Bakker, W.; van der Werff, H.; van der Meer, F. Determining Smile And Keystone Of Lab Hyperspectral Line Cameras. In Proceedings of the 2019 10th Workshop on Hyperspectral Imaging and Signal Processing: Evolution in Remote Sensing (WHISPERS), Amsterdam, The Netherlands, 24–26 September 2019; pp. 1–5. [CrossRef]
35. Kopačková, V.; Koucká, L. Mineral mapping based on automatic detection of multiple absorption features. *EARSeL EProc.* **2014**, *13*, 95–99. [CrossRef]
36. Therien, C. Welcome to the PySptools Documentation. 2022. Available online: https://pysptools.sourceforge.io/ (accessed on 24 November 2023).
37. Boggs, T. Spectral Python (SPy) User Guide. 2020. Available online: https://www.spectralpython.net/user_guide.html (accessed on 24 November 2023).
38. Grizonnet, M.; Michel, J.; Poughon, V.; Inglada, J.; Savinaud, M.; Cresson, R. Orfeo ToolBox: Open source processing of remote sensing images. *Open Geospat. Data Softw. Stand.* **2017**, *2*, 15. [CrossRef]
39. van der Linden, S.; Rabe, A.; Jakimow, B.; Thiel, F.; Cooper, S.; Okujeni, A.; Hostert, P. Integrating Imaging Spectroscopy and GIS—Free and Open Source Image Analysis in QGIS with the EnMAP-Box 3. In Proceedings of the OSA Optical Sensors and Sensing Congress 2021 (AIS, FTS, HISE, SENSORS, ES), Washington, DC, USA, 19–23 July 2021; Optica Publishing Group: Washington, DC, USA, 2021; p. HF4E.2. [CrossRef]
40. van Rossum, G.; Fridman, L. Why Python 3.11 Is So Fast. 2022. Interview on YouTube. Available online: https://youtu.be/TLhRuZ9cJWc (accessed on 31 July 2024).
41. Modular Inc. Mojo—A New Programming Language. 2023. Available online: https://www.modular.com/mojo (accessed on 24 November 2023).
42. Bakker, W.H. wimhbakker/hyppy: HypPy3. *Zenodo* **2024**. [CrossRef]