Polycomp Documentation

Release 1.0

Maurizio Tomasi

Oct 08, 2017

Contents

Contents:

CHAPTER 1

How to get the latest Polycomp version

To obtain the latest version of Polycomp, first be sure to have an updated version of the `libpolycomp` library. It is available at the link https://github.com/ziotom78/libpolycomp.

The source code for Polycomp is available at https://github.com/ziotom78/polycomp: download and install it using `git` (https://git-scm.com/). Refer to that page for any prerequisite. The typical sequence of commands used to install `polycomp` is the following:

```
git clone https://github.com/ziotom78/polycomp
cd polycomp
python3 setup.py build
```

If you want to install `polycomp` and the Python bindings to `libpolycomp`, you must run the command `install`:

```
python3 setup.py install
```

either as a super-user or using `sudo`.

Requisites

Polycomp requires the following tools:

- Python (both version 2 and 3 are fine);
- *click* (http://click.pocoo.org/5/), automatically installed by *setup.py*;
- *numpy* (http://www.numpy.org/), automatically installed by *setup.py*;
- Either *astropy* (version 0.4 or greater, http://www.astropy.org/, the preferred solution) or *pyfits* (http://www.stsci.edu/institute/software_hardware/pyfits). (The *setup.py* script installs Astropy.)

CHAPTER 2

An introduction to Polycomp

`polycomp` is a Python program that relies on the *libpolycomp* library (https://github.com/ziotom78/libpolycomp) to compress one-dimensional datasets in FITS format. It has been optimized for datasets produced by astronomical experiments where large FITS tables store time-ordered measurements.

The program takes as input tables in FITS files and assembles them into files with extension `.pcomp`. The latters are FITS files in disguise, where each table taken from the input files is compressed using one of the following compression schemas:

Run-length encoding This works extremely well for columns containing long sequences of repeated integer values, such as quality flags.

Differenced run-length encoding This is a variant of the preceding scheme, and it works with integer numbers which increase regularly. A typical example is the time recorded by a digital clock.

Quantization This lossy compression scheme reduces the number of bits used for storing floating-point numbers. It is useful for compressing sequences of numbers measured by digital instruments, if the number of bits of the raw measurement is significantly lower than the standard size of floating-point numbers (32-bit/64-bit).

Polynomial compression This compression works for smooth, noiseless data. It is a lossy compression scheme. The program allows to set an upper bound to the compression error.

Zlib This widely used compression is implemented using the `zlib` library (http://www.zlib.net/).

Bzip2 Another widely used compression scheme, implemented using the `bzip2` library (http://www.bzip.org/).

`polycomp` is a command-line program which can either compress or decompress data. In the following sections we illustrate how to compress a set of FITS files into one single `.pcomp` file, and then we show how to decompress it.

Compressing files using polycomp

To compress tables from one or more FITS files, the user must prepare a *configuration file* which specifies where to look for the tables and which kind of compression apply to each of them. Such configuration files are parsed using the widely used `configparse` Python library: the syntax of such files is described in the standard Python documentation, available at https://docs.python.org/3/library/configparser.html.

Let's consider as an example how to compress the TOI files available on the Planck Legacy Archive (http://www.cosmos.esa.int/web/planck/pla). Such TOIs contain the time-ordered information acquired by the two instruments onboard the spacecraft, LFI and HFI. In order to cut download times, we consider the smallest TOIs in the database: those produced by the radiometer named LFI-27M. There are more than one kind of TOIs: we concentrate on the two most useful ones, the differenced TOIs (containing the actual measurements of the radiometer) and the pointing TOIs (containing the direction of view as a function of time).

Download the TOIs at the following addresses:

- http://pla.esac.esa.int/pla-sl/data-action?TIMELINE.TIMELINE_OID=36063 (differenced scientific data, 96 MB, save it into file `sci.fits`)
- http://pla.esac.esa.int/pla-sl/data-action?TIMELINE.TIMELINE_OID=2062650 (pointings, 223 MB, save it into file `ptg.fits`)

Such TOIs contain the time-ordered information for all the 30 GHz radiometers (LFI-27M, LFI-27S, LFI-28M, LFI-28S): each HDU contains information about just one radiometer. We are interested only in LFI-27M, so we must tell `polycomp` which data to extract from both files.

Create a text file named `pcomp_LFI27M.conf` with the following content:

```
[polycomp]
tables = obt_time, theta, phi, psi, diff, flags

[obt_time]
file = sci.fits
hdu = 1
column = OBT
compression = diffrle
datatype = int64

[theta]
file = ptg.fits
hdu = LFI27M
column = THETA
compression = polynomial
num_of_coefficients = 8
samples_per_chunk = 80
max_error = 4.8e-6
use_chebyshev = True

[phi]
file = ptg.fits
hdu = LFI27M
column = PHI
compression = polynomial
num_of_coefficients = 8
samples_per_chunk = 80
max_error = 4.8e-6
use_chebyshev = True

[psi]
file = ptg.fits
hdu = LFI27M
column = PSI
compression = polynomial
num_of_coefficients = 8
samples_per_chunk = 80
max_error = 4.8e-6
use_chebyshev = True
```

```
[diff]
file = sci.fits
hdu = LFI27M
column = LFI27M
compression = quantization
bits_per_sample = 20

[flags]
file = sci.fits
hdu = LFI27M
column = FLAG
compression = rle
datatype = int8
```

This file describes the way input data will be compressed by `polycomp`. Run the program with the following syntax:

```
polycomp.py compress pcomp_LFI27M.conf compressed.pcomp
```

This command will produce a file named `compressed.pcomp`, which contains the six compressed columns of data specified in the configuration file. The file format used for `compressed.pcomp` is based on the FITS standard, and you can therefore use any FITS library/program to access its data. (Of course, to actually decompress the data in it you must use `libpolycomp`.)

Decompressing files using polycomp

Decompression is considerably simpler than compression, as it does not require to prepare a configuration file. You have to specify the input `.pcomp` file and the output FITS file, as in the following example:

```
polycomp.py decompress compressed.pcomp decompressed.fits
```

By default, `polycomp` will save every column of data in a separated HDU file within `decompressed.fits`. If all the columns in `compressed.pcomp` are of the same length, you can use the `--one-hdu` flag to save everything in one HDU:

```
polycomp.py decompress --one-hdu compressed.pcomp decompressed.fits
```

CHAPTER 3

Writing a configuration file

In this section we describe how to write a configuration file which tells Polycomp which data to compress, and how.

Note for Python developers: Polycomp configuration files are parsed using Python's *configparse* library. Refer to its documentation for more information about all the facilities provided by this format.

Basic syntax

Empty lines and lines starting with `#` are ignored.

Polycomp configuration files are divided in sections, each marked by its name enclosed in square brackets, like in the following example:

```
[section 1]
# Here are the contents of "section 1"

[section 2]
# Et cetera
```

Within each section, the file is expected to contain a sequence of key/value pairs, as in the following example:

```
cmb_temperature = 2.7
speed_of_light = 3.0e8
```

Strings must be specified without single/double quotes:

```
file_path = /data/my_test_data.fits
```

Specifying which data to compress

Every configuration file must at least contain one section named `polycomp`. This section should contain the key `tables`, which specifies a comma-separated list of tables that will be included in the compressed `.pcomp` file

generated by the program. For every table specified here there must be a section (either before or after this one) with the same name, where details about the table are to be provided.

Here is an example:

```
[polycomp]
tables = time, temperature, pressure

[time]
# Here we specify where to take time data, and how to compress them

[temperature]
# Ditto for the temperature...

[pressure]
# ...and for the pressure
```

In the next sections we are going to explain what should every "table section" contain.

Where to take the data to compress

In each table section the following key/value pairs must be present:

- The *file* key specifies the path to the FITS file containing the data to be compressed.
- The *hdu* key specifies the number/name of the HDU within the FITS file (the first HDU is 1).
- The *column* key specifies the number/name of the column in the HDU (the first column is 1).

Here is an example:

```
file = /opt/data/experiment_1.fits
hdu = 1
column = TIME
```

How to compress the data

The *compression* key must be present in each table section. It contains a string which identifies the compression algorithm to use, and it can be one of the following:

Value	Algorithm
`none`	No compression
`rle`	Run-Length Encoding (RLE)
`diffrle`	Differential RLE
`quantization`	Quantization of floating-point values
`polynomial`	Polynomial compression
`zlib`	Zlib-based compression
`bzip2`	Bzip2-based compression

The `none`, `rle` and `diffrle` compression algorithms do not require other key/value pairs. For all the other cases, they are presented in the next subsections.

Quantization parameters

The only parameter required for this kind of compression is `bits_per_sample`, which specifies the number of bits to be used with each sample. Typical values are integers less than 32 or 64 bits, depending on the width of the floating-point type used in the input data.

Polynomial compression parameters

There are a number of key/value pairs that are understood when using this algorithm. Not every pair is required; in a handful of cases, Polycomp can provide a default value.

Key	Default value
`num_of_coefficients`	(Required)
`samples_per_chunk`	(Required)
`max_error`	(Required)
`use_chebyshev`	`True`
`period`	If not specified, the input data will be assumed not to be periodic.
`no_smart_optimization`	False
`opt_delta_coeffs`	1
`opt_delta_samples`	1
`opt_max_num_of_iterations`	0 (no upper limit)

The meaning of the parameters is the following:

- `num_of_coefficients` specifies the number of coefficients of the fitting polynomial, i.e., `deg p(x) + 1`. The best value for this parameter depends heavily on the input data to be compressed.
- `samples_per_chunk` specifies the number of samples in each chunk. This number must always be greater than `num_of_coefficients`.
- If `use_chebyshev` is set to `False`, the so-called "simple compression algorithm" will be used. In some situations the code might run faster, but it can produce significantly worse compression ratios.
- If the input data are periodic, `period` should be set to their period (e.g., 2π for angles measured in radians). The default is not to assume the input data periodic.

The remaining parameters (`no_smart_optimization`, `opt_delta_coeffs`, `opt_delta_samples`, and `opt_max_num_of_iterations`) are used when the user wants to search the best possible configuration for the polynomial compressor.

Zlib/Bzip2 parameters

CHAPTER 4

Optimizing polynomial compression parameters

Polycomp implements a number of tools to simplify the choice of the parameters for polynomial compression. The parameters are N_{chunk}, the number of samples per chunk, and $\deg p(x) + 1$, the number of coefficients in the interpolating polynomial. Polycomp's strategy to optimize these parameters is to test a number of configuration and pick the one with the best compression ratio C_r.

Two algorithms are implemented:

- A naive algorithm that scans rectangular regions of the parameter space $N_{\text{chunk}} \times (\deg p(x) + 1)$;
- A simplex-downhill algorithm that hunts for local minima in the parameter space.

Scanning rectangular regions of the parameter space

To find the best values for N_{chunk} and $(\deg p(x) + 1)$ within a rectangular region of the parameter plane, the user can specify the values to be checked using the usual parameters `num_of_coefficients` and `samples_per_chunk`. Instead of specifying only one value, a set of values can be specified using the following syntax:

- Ranges can be specified using the syntax `NN-MM` or `NN:MM`;
- Intervals can be specified using the syntax `NN:D:MM`, where `D` is the interval;
- Multiple values and ranges can be concatenated, using commas (`,`) as separators.

Simplex-downhill strategy

CHAPTER 5

Polycomp file format

We present here a detailed description of the compressed files written by Polycomp.

Polycomp files are FITS files where each 1-D datastream is saved in its own HDU. A number of FITS keyword defined here are used to store details about the datastream, such as the compression algorithm and other information needed for the decompression. It is possible to access Polycomp files using any FITS library. Notable examples are cfitsio and Astropy (through the `io.fits` module).

Keywords

Any Polycomp file contains one or more binary table HDUs. Each HDU contains the data required to decompress one 1-D table. The order of the HDUs is the same as the order specified in the Polycomp configuration file used for the compression.

The following keyword are defined in each of the table HDUs:

Keyword	Type	Meaning
`PCSRCTP`	String	NumPy type of the input data
`PCCOMPR`	String	Kind of compression algorithm used
`PCNUMSA`	Integer	Number of uncompressed samples
`PCNUMSA`	Integer	Number of uncompressed samples
`PCUNCSZ`	Integer	Size of the uncompressed samples, in bytes
`PCCOMSZ`	Integer	Size of the compressed samples, in bytes
`PCTIME`	Float	Time needed to compress the data
`PCCR`	Float	Compression ratio

The allowed strings for `PCCOMPR` are the following:

- `none`: no compression;
- `rle`: Run-Length Encoding;
- `diffrle`: Differenced Run-Length Encoding;
- `quantization`: Floating-point quantization;

- `polynomial`: Polynomial compression (with or without the Chebyshev step);
- `zlib`: zlib-based compression;
- `bzip2`: bzip2-based compression.

If the algorithm is `quantization`, the following keywords are saved in the HDU header as well:

Keyword	Type	Meaning
`PCELEMSZ`	Integer	Number of bits per uncompressed sample
`PCBITSPS`	Integer	Number of bits per compressed sample
`PCNORM`	Float	Normalization factor
`PCOFS`	Float	Offset

Table data

Table data is saved in one fixed-size column (with one notable exception, see below). The type of this column depends on the input data type and/or the compression algorithm:

Compression	Column type
`none`	Same as input data
`rle`	Same as input data
`diffrle`	Same as input data
`quantization`	8-bit integer
`polynomial`	8-bit integer (but see below)
`zlib`	8-bit integer
`bzip2`	8-bit integer

Debug-mode for polynomial compression

Due to its relative complexity, polynomial compression can be saved using a special debug mode. In this mode, instead of coding the compressed stream as a raw sequence of 8-bit integers, the compressor saves information about each chunk separately. This mode is not as efficient as the default mode, but it allows to understand the compressor's performance more easily.

If a data stream is compressed using polynomial compression in debug mode, the keyword `PCDEBUG` in the HDU header is set to 1, and the following columns are saved:

Name	Type	Row size	Description
`ISCOMPR`	Logical	1	True if this chunk has been compressed
`CKLEN`	Integer	1	Length of this chunk
`UNCOMPR`	Double	Variable	If `ISCOMPR` is False, the uncompressed samples
`POLY`	Double	Variable	If `ISCOMPR` is True, the polynomial coefficients
`CHMASK`	8-bit integer	Variable	If `ISCOMPR` is True, the Chebyshev bit mask
`CHEBY`	Double	Variable	If `ISCOMPR` is True, the Chebyshev coefficients

The length of the rows in variable-length columns is the following:

- `UNCOMPR`: a number of elements equal to `CKLEN`;
- `POLY`: equal to $\deg p(x) + 1$;
- `CHMASK`: equal to $\lceil N + 1 \rceil$, where N is `CKLEN`;
- `CHEBY`: equal to the number of bits in `CHMASK` that are equal to 1.

CHAPTER 6

Indices and tables

- genindex
- modindex
- search