ADASS 2016, Trieste (Italy)

Compression of smooth data tables using Polycomp

Maurizio Tomasi (Università degli Studi di Milano, Italy)

Data compression is increasingly important in astrophysics, as the amount of data acquired by modern experiments often needs hundreds of terabytes for the storage of raw data. Here I present a few usage cases of "polycomp", a C/Python library to compress smooth one-dimensional data whose error is either zero or negligible. One of the algorithms implemented by "polycomp" combines the advantages of polynomial least-squares fitting and the properties of the discrete Chebyshev transform. This algorithm can lead to compression ratios larger than 10 in a number of realistic cases. I will show a few examples of datasets that can be easily compressed using this approach, namely: (1) spacecraft attitude information, and (2) timelines of pointing information for a realistic all-sky survey experiment.

Polycomp in a nutshell

Polycomp is a Python program and a set of Python bindings to Libpolycomp, a C library, to implement a number of compression algorithms useful for compressing one-dimensional time series. Both Libpolycomp and Polycomp are available under a permissive MIT license.

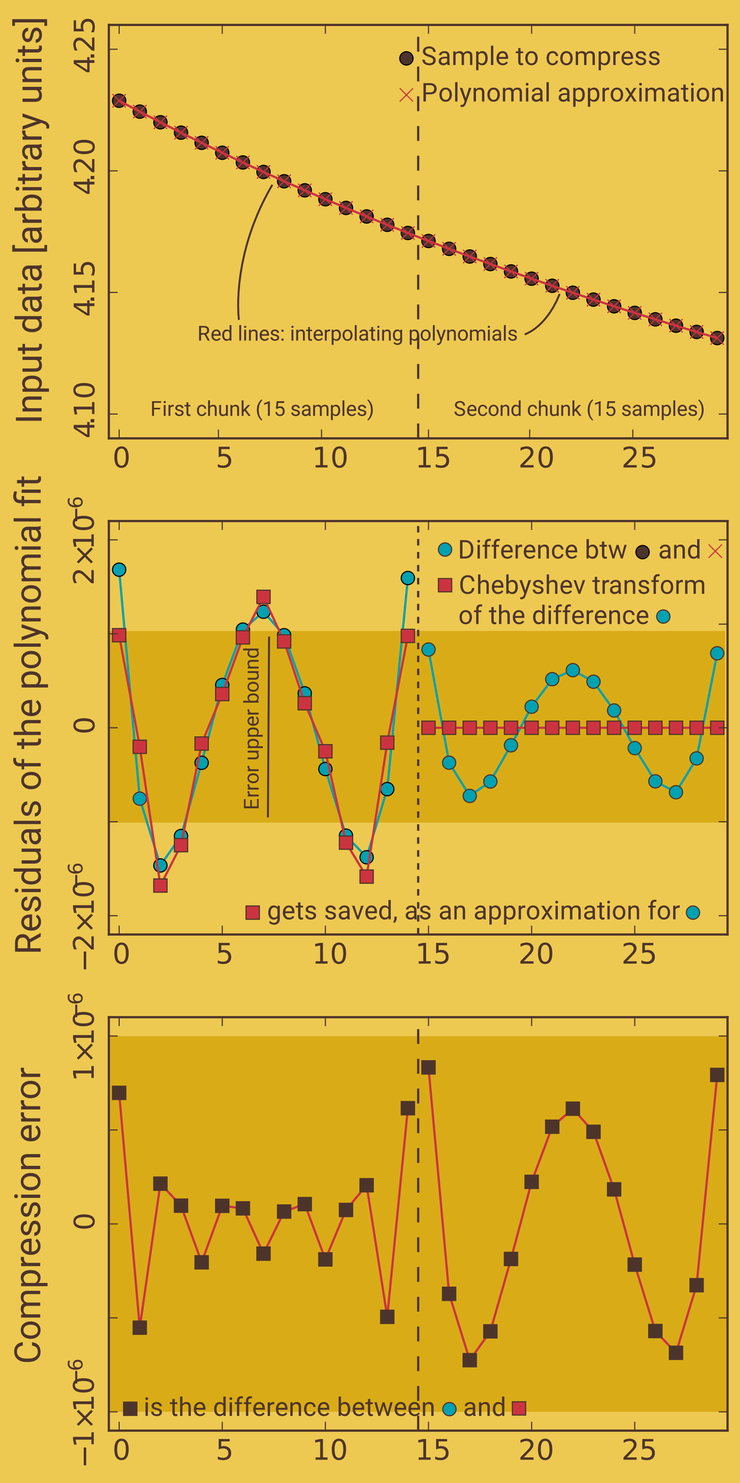
The C library implements a number of compression schemes, all described in Tomasi (2016). Most of the compression algorithms are mainly meant to compress timelines where the amount of noise is negligible (e.g., pointing information, flags, time information…). Both the C and Python libraries compress data using memory buffers. The Python program is able to save compressed data to files, using FITS files as containers. The structure of these FITS files are fully documented.

Polynomial interpolation

The polynomial interpolation algorithm is a quite sophisticated lossy compression scheme implemented by Polycomp. It implements a mixture of least-squares fit and Fourier transforms to achieve good compression ratios in a fairly large number of situations. The algorithm works as follows:

1. Split the sequence of samples in chunks of a given size;
2. Approximate the samples in each chunk with a polynomial of a given degree;
3. Compute the residuals of the approximation;
4. If the residuals are above some threshold, approximate the residuals with a filtered Chebyshev transform;
5. Save the polynomial coefficients and the Chebyshev transform.

A «filtered Chebyshev transform» is a Chebyshev transform where the smallest coefficients are set to zero and not saved to disk. The number of zeroed coefficients is chosen in order for the error not to be greater than a user-specified upper bound. The plot below explains how the algorithm works:



The first step is to split the dataset in chunks. In this case, I chose to use 15 samples per chunk. Within each chunk, the samples are approximated with a polynomial of some given degree (the same degree applies to all the chunks).

The discrepancy between the samples and the polynomial fit are measured against a user-provided error threshold (10^{-6}). Since the samples in the first chunk does not satisfy the bound, Polycomp computes a filtered Chebyshev transform of the residuals.

Polycomp saves the coefficients of the interpolating polynomials, as well as the nonzero Chebyshev coefficients of the residuals. For both chunks, the residual compression error is less than the upper bound (10^{-6}), as expected.

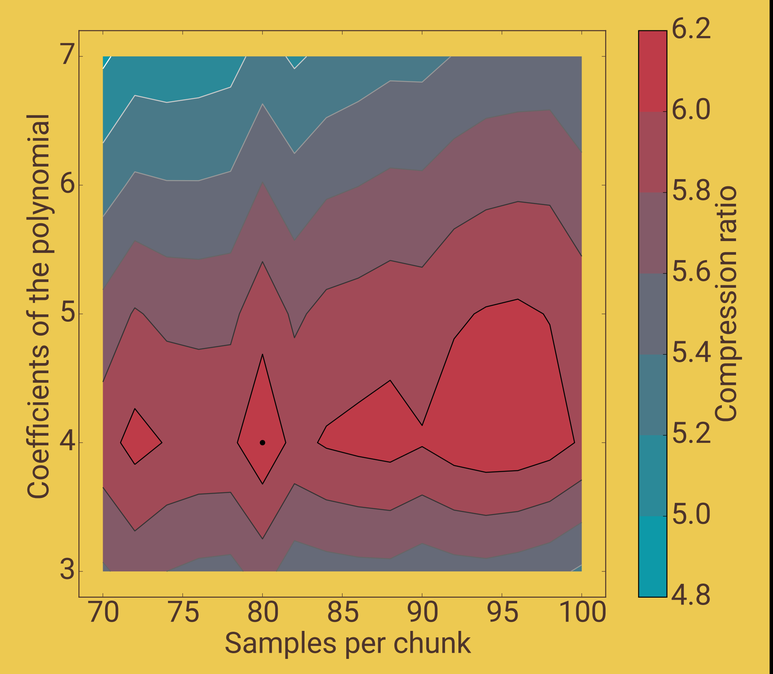
Example #1: spacecraft speed

I show a simple example of the application of Polycomp to the time series of the velocity vector of the ESA Planck spacecraft. These can be obtained using the JPL's HORIZONS system (http://ssd.jpl.nasa.gov/?horizons). This kind of dataset is useful to compute the amount of Doppler signal caused by the motion of the spacecraft with respect to the Sun.

The input file is a 70 MB FITS file with one binary table HDU containing four columns: the time and the three cartesian components of the velocity, sampled once every minute. It contains ~2 million rows, which corresponds to almost 4.5 years of data, saved in a 69.9 MB FITS file.

I used the polynomial compression algorithm (see the panel on the left) to compress the four columns, Polycomp ran the compression several times, varying each time either the degree of the interpolating polynomial or the number of samples in each chunk, and picking up the combination which produced the best compression ratio. The result was saved in a compressed 10.5 MB FITS file, with a compression ratio equal to 6.6.

The plot on the right shows the result of the optimization for the X component of the velocity vector: the dot at (80, 4) is the best configuration.



Example #2: spacecraft pointing information

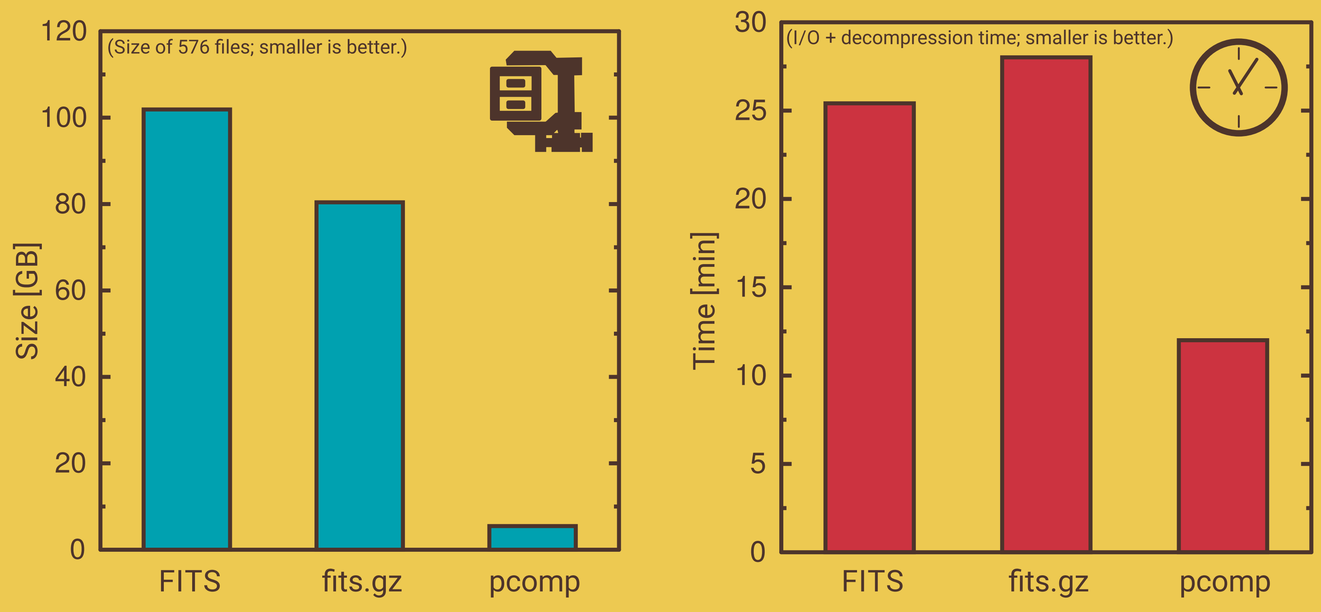
I produced a set of pointing information for an ideal spacecraft for CMB measurements, observing the sky through a telescope and implementing polarization-sensitive detectors. I used the so-called «CORE-like» scanning strategy provided as an example of the TOAST package (https://github.com/hpc4cmb/toast).

The pointing information was saved in 576 FITS files, with an overall size of 102 GB. Each FITS file contained one binary table HDU with the following columns: (1) time (double precision number), (2) latitude, (3) longitude, (4) polarization angle, (5) sky signal, (6) Doppler signal, (7) noise signal, (8) overall signal, sum of the sky, the Doppler effect, and the noise, (9) flags. All the columns but the last one contain double-precision numbers; the «flags» column contains 64-bit integer.

I used Polycomp to compress the time, latitude, longitude, and polarization columns using the «polynomial compression» (see panel on the left), the four columns containing the signal using 24-bit quantization, and the flags using RLE compression. I searched for the optimal set of parameters for the polynomial compression on each of the 576 files, using as upper bound for the compression error 1 μs (time) and 1 arcsec (angles).

Disk occupation and decompression speed

The set of 576 files was compressed using gzip and Polycomp itself. The size of each file set is shown in the plot on the left, while the time required to read and decompress sequentially them is shown on the right. Note how Polycomp beats gzip both in terms of compression ratio and read+decompression time.



Compression error

In order to measure the impact of the compression error, I calculated the inverse of the pixel condition number (Eq. 10 in Kurki-Suonio et al., 2009). This measures the ability to reconstruct the polarization parameters for each pixel in a sky map by means of one number in the range [0, 1] per pixel. I used a Healpix map (Górski et al., 2005) with NSIDE=1024, containing roughly 10^7 pixels. I redid the calculation using as input each of the three sets of file above, and I measured the discrepancies. Since the gzip compression is lossless, using the gzipped files instead of the uncompressed ones resulted in no change. Using Polycomp, the inverse condition numbers had a mean absolute error of ~10^{-8}, with an upper limit of 0.01 (0.002 ‰ of the number of pixels). Such errors are caused by the polynomial compression and can be reduced by reducing the upper bound on the compression error.

References

Polycomp: efficient and configurable compression of astronomical timelines, Tomasi M., A&C, vol. 16 (2016), pages 88–98, http://dx.doi.org/10.1016/j.ascom.2016.04.004
Destriping CMB temperature and polarization maps, Kurki-Suonio, H. et al., A&A, vol. 506 (2009), pages 1511–1539, http://dx.doi.org/10.1051/0004-6361/200912361
HEALPix: a framework for high-resolution discretization and fast analysis of data distributed on a sphere, Górski et al., ApJ, vol. 622 (2005), pages 759–771, http://dx.doi.org/10.1086/427976
Libpolycomp (C library): https://github.com/ziotom78/libpolycomp [ascl:1604.002]
Polycomp (Python bindings and stand-alone executable): https://github.com/ziotom78/polycomp