https://doi.org/10.5194/gmd-2020-239
Preprint. Discussion started: 30 July 2020

Geoscientific Model Development
Discussions

A note on precision-preserving compression of scientific data

Rostislav Kouznetsov[1,2]

[1]Finnish Meteorological Institute, Helsinki, Finland
[2]Obukhov Institute for Atmospheric Physics, Moscow, Russia

Correspondence: Rostislav Kouznetsov (Rostislav.Kouznetsov@fmi.fi)

Abstract. Lossy compression of scientific data arrays is a powerful tool to save network bandwidth and storage space. Properly applied lossy compression can reduce the size of a dataset by orders of magnitude keeping all essential information, whereas a wrong choice of lossy compression parameters leads to the loss of valuable data. The paper considers statistical properties of several lossy compression methods implemented in "NetCDF operators" (NCO), a popular tool for handling and transformation of numerical data in NetCDF format. We compare the effects of imprecisions and artifacts resulting from use of a lossy compression of floating-point data arrays. In particular, we show that a popular Bit Grooming algorithm (default in NCO) has sub-optimal accuracy and produces substantial artifacts in multipoint statistics. We suggest a simple implementation of two algorithms that are free from these artifacts and have twice higher precision. Besides that, we suggest a way to rectify the data already processed with Bit Grooming.

The algorithm has been contributed to NCO mainstream. The supplementary material contains the implementation of the algorithm in Python 3.

1 Introduction

Floating-point formats used for scientific data are implemented in most of computer systems according to the IEEE 754-1985 standard (ANSI/IEEE, 1985). The standard offers two formats: single-precision (also known as float) and double-precision (or just double). The data in these formats occupy 32 and 64 bits per value in memory and have precisions of approximately 7 and 16 significant decimal figures.

The variety of precisions needed in practice is much larger. Many scientific datasets have only few (or even few tens of) percent accuracy and thus require much less than 7 decimals. As a result application of general-purpose lossless compression algorithms to such datasets does not lead to substantial reduction of the dataset size since non-significant bits of the values contain arbitrary numbers with high entropy, that are difficult to compress. Setting these bits to a prescribed value (trimming precision) substantially reduces the entropy of the dataset making lossless compression algorithms much more efficient. A combination of precision trimming and a lossless compression constitutes an efficient method of lossy compression with well controlled loss of precision.

Zender (2016) implemented the precision-trimming in a versatile data-processing tool-set called "NetCDF operators" (NCO http://nco.sourceforge.net, last accessed on July 14, 2020), enabling the internal data compression features of the NetCDF4 format (https://www.unidata.ucar.edu/software/netcdf last accessed on July 14, 2020) work efficiently. The tool-set allows a

https://doi.org/10.5194/gmd-2020-239
Preprint. Discussion started: 30 July 2020

user to chose the needed number of decimals for each variable in a file, so the precision trimming can be applied flexibly depending on the nature of the data. It has been quickly noticed hat a simple implementation of a precision trimming by setting non-significant bits to zero (bit shaving) introduces undesirable bias to the data. As a way to cope with that bias Zender (2016) introduced a Bit Grooming algorithm that alternately shaves (to zero) and sets (to one) the least significant bits of consecutive values. A detailed review of the compression methods used for scientific data as well as the analysis of the influence of the compression parameters on compression performance can be found in the paper by Zender (2016) and references therein.

In the current note we consider several precision-trimming algorithms and inaccuracies they introduce. Our analysis revealed a major distortion two-point structure function of fields, caused by Bit Grooming. Therefore we suggest an implementation of mantissa-rounding that provides twice higher accuracy of the data representation than Bit Grooming with the same number of data bits. Besides that, a method of recovering the Bit-Groomed data is suggested.

The paper organized as follows. The next section formulates the concept of precision trimming and describes various methods for it and their implementation. Sec. 3 analyses errors introduced by the precision trimming with various parameters. Sec. 4 illustrates the effect of precision-trimming on two synthetic datasets. Conclusions are summarized at the end of the paper.

2 Precision-trimming methods

Hereafter we will consider single-precision floating-point representation, although all the conclusions can be extrapolated on arbitrary-precision numbers. A computer representation of a single-precision number according to ANSI/IEEE (1985) consists of 32 bits: 1 bit for sign, 8 bits for exponent and 23 bits for mantissa (also called “fraction”), having the most-significant bit (MSB) of mantissa implicit 1, allowing for 24-bit resolution of mantissa. M bits of mantissa allow for distinguishing any two numbers a and b if

$$2\frac{|a-b|}{|a|+|b|} > 2^{-M}, \tag{1}$$

therefore the representation provides $2^{-24} \simeq 6 \cdot 10^{-8}$ relative precision, or about 7 decimals.

If the data have smaller precision, the least-significant bits (LSBs) of mantissa contain arbitrary, often chaotic information, which makes lossless-compression algorithms inefficient. If one trims the precision, i.e. replaces those bits with a pre-defined value, the efficiency of compression improves. Therefore the problem of trimming LSBs can be formulated as follows: given an array of floating-point numbers set transform it so, that the resulting values have all but N LSBs of their mantissa set to a prescribed combination, while the difference from the original array is minimal.

Hereafter we will use a term “keep-bits” for N most-significant bits of mantissa, that we let to store the information from the original values, and “tail bits” the remaining bits of a mantissa that we are going to set to a prescribed value. The resulting precision is characterized by the number of bits of mantissa that we are going to keep. The precision of the resulting values is given by Eq. 1.

The following methods have been considered:

– *shave* – set all tail bits to zero

https://doi.org/10.5194/gmd-2020-239
Preprint. Discussion started: 30 July 2020

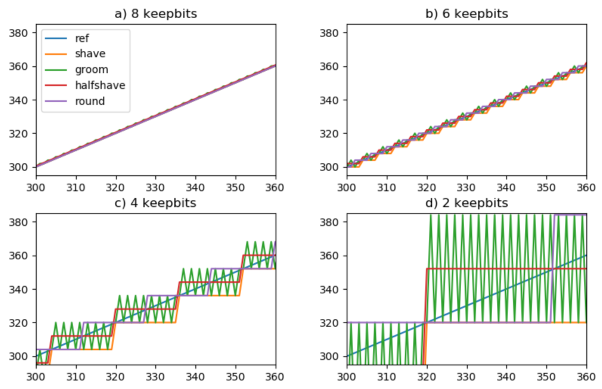

Figure 1. Trimmed values as a function of original values when trimmed to different number of keep-bits. The lines correspond to the different methods

– *set* – set all tail bits to ones

– *groom* – set all tail bits to ones or to zero in alternate order for every next element of the array

– *halfshave* – set all tail bits to zero except for the most significant one of them, which gets set to one

– *round* – rounding of mantissa to the nearest value that has zero tail bits. Note that this rounding affects keep-bits, and might affect the order bits as well

The former three of them have been implemented in NCO some years ago and described well by Zender (2016). *Halfshave* and *round* have been recently introduced into NCO. *Halfshave* is trivial to implement by applying an integer bit-mask to a floating-pint number: first one applies the *shave* bit-mask and then sets MSB of tail-bits to 1.

Implementation of rounding is somewhat less trivial. Since rounding might affect keep-bits, and even the exponent bits, implementing it with bitwise operations is difficult. However combining bit-masking with floating-point operations makes it simple. Consider the original value u. It can be expressed via the result of bit-shaving it v:

$$u = v + \epsilon, \tag{2}$$

where ϵ is their difference. One can get a rounded value of v by bit-shaving $u + \epsilon = 2u - v$. Indeed, if adding ϵ to v changes the keep-bits or exponent, v should be rounded away from zero and vice-versa: if adding ϵ to v does not change keep-bits and exponent, v should be rounded towards zero. In both cases bit-shaving of $2u - v$ results in rounding towards the nearest value with zero tail-bits.

https://doi.org/10.5194/gmd-2020-239
Preprint. Discussion started: 30 July 2020

mantissa size bit	keep-bits	discretisation level increment as a fraction of the value			max relative error *shave*, *set*, *groom*	max relative error *round*, *halfshave*
1	0	0.5	…	1	100%	50%
2	1	0.25	…	0.5	50%	25%
3	2	0.125	…	0.25	25%	12.5%
4	3	0.0625	…	0.125	12.5%	6.25%
5	4	0.03125	…	0.0625	6.25%	3.1%
6	5	0.015625	…	0.03125	3.1%	1.6%
7	6	0.0078125	…	0.015625	1.6%	0.8%
8	7	0.00390625	…	0.0078125	0.8%	0.5%
9	8	0.001953125	…	0.00390625	0.5%	0.25%

Table 1. The discretisation level increment for different number of mantissa bits kept and maximum relative error introduced by trimming the precision with various methods

3 Quantification of errors

To characterize the trimming we will use the number of explicit bits kept in mantissa of a floating point number. Therefore "bits = 1" means that the mantissa keeps 1 implicit and 1 explicit bits for the value, and the remaining 22 bits have been reset to a prescribed combination.

When rounding to a fixed quantum (e.g. to integer), the magnitude of maximal introduced error is a function of the quantum. The error is one quantum for one-sided rounding (up, down, towards zero, away from zero), and half of the quantum for rounding to the nearest value. With precision-preserving compression, the quantum depends on the value itself. For a value v discretized with M bits of mantissa kept ($M-1$ keep-bits) the quantum is the value of the least-significant bit of keep-bits, i.e. at least $|2^{1-M}v|$, and at most $|2^{-M}v|$ depending on how far v is from the maximum power of 2 that is less than v. Note that the quantisation levels for the *halfshave* method are in mid-points of the intervals between the levels for other methods.

The *shave* and *set* methods of above are similar to rounding towards zero and away from zero correspondingly. They introduce an error of at most full quantum to each value with mean absolute error of a half-quantum. The *groom* method being alternate *shave* and *set* has the same single-value statistics as *shave* and *set*. The *round* method takes into account the value of the MSB of tail bits, an therefore introduces an error of at most half-quantum with mean absolute error of a quarter of a quantum. Same applies to the he *halfshave* method, since its levels are midpoints of the range controlled by the tail-bits of the original value. Note that for the mean-absolute error estimate we assume that the tail bits have equal chance to be 0 or 1.

The margins for an error introduced by the considered methods for different number of mantissa bits are summarized in Table. 1. As one can see, the *round* and *halfshave* method allow for twice smaller error for the same number of keep-bits or one less keep-bit to guarantee the same accuracy.

The results of applying a precision trimming with various number of keep-bits are illustrated in Fig. 1, where the 1:1 line is marked as "ref". The *set* method is not shown, to reduce number of lines in the panels. For the shown range and keep-bits

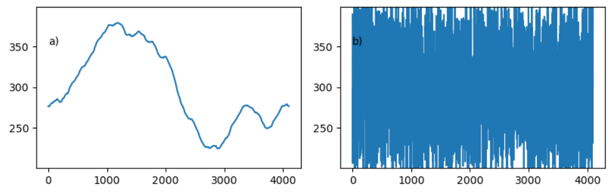

Figure 2. Two 1D signals used for illustration

	2 keep-bit	4 keep-bit	6 keep-bit	8 keep-bit
shave	0.09523	0.02625	0.00683	0.00177
groom	0.11262	0.02776	0.00704	0.00177
halfshave	0.05617	0.01333	0.00358	0.00089
round	0.05462	0.01440	0.00342	0.00087
groomhalf	0.05617	0.01333	0.00358	0.00089
groomav	0.05607	0.01319	0.00345	0.00079

Table 2. NRMSE of the signal in Fig. 2a after trimming precision

of 8, 4, 6, and 2 the discrete levels are separated by 0.5, 2, 8, and 32 correspondingly. Note that the levels of *halfshave* have half-discrete offset with respect to others. For 8 keep-bits (Fig. 1a) in the scale of the plots the distortion is almost invisible, whereas for 2 keep bits the difference is well seen.

As it was pointed by Zender (2016) the *shave* method introduces a bias towards zero, which might be unwanted in some applications (*set* introduces the opposite bias). The idea of Bit Grooming (*groom*) is to combine these two biased trimming procedures to get on average unbiased fields. With bit grooming, the quantization of a value in an array depends not only on the value itself, but also on its index in the array. While fixing the mean bias, the Bit Grooming introduces an additional oscillating component to the fields that affects multipoint point statistics. For instance, the absolute difference between two values with even and odd indices will get a positive bias in average, while the absolute difference between two points with both even (or both odd) indices will stay unbiased.

As a way to compensate the artifacts of bit grooming without introducing additional bias one could apply moving average (C. Zender, private communication). This procedure, however still affects two-point statistics and might be more damaging than Bit Grooming. One can note that the result of applying the *halfshave* procedure with the same number of keep-bits to a bit-groomed field is equivalent to the applying it to the original field. Therefore *halfshave* can be considered as a way to half the error and to remove the artifacts of Bit Grooming.

https://doi.org/10.5194/gmd-2020-239
Preprint. Discussion started: 30 July 2020

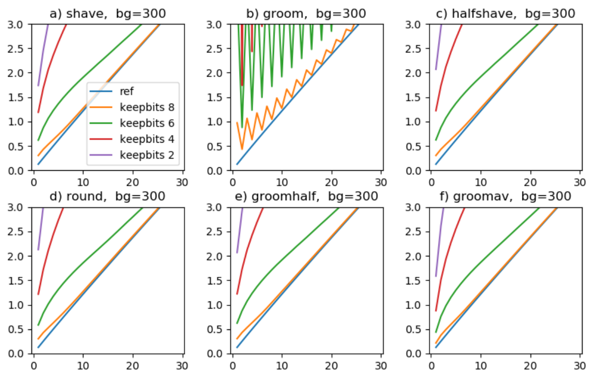

Figure 3. The structure functions of a signal shown in Fig. 2a, processed with different precision-trimming

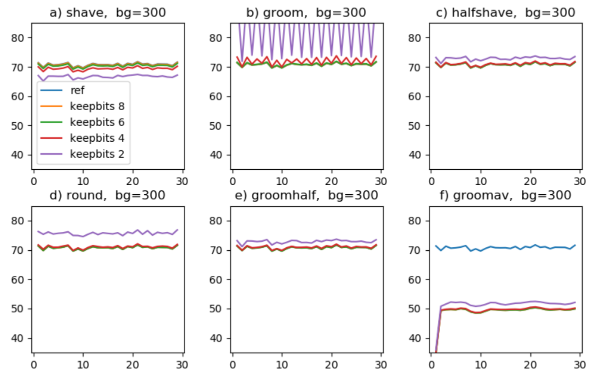

Figure 4. The structure functions of a signal shown in Fig. 2b, processed with different precision-trimming

4 Examples

Consider an array of N floating-point numbers u_i and its precision-trimmed version v_i. To illustrate the performance of the algorithms we will consider normalised root-mean-square error (NRMSE) introduced by a precision-trimming

$$\mathrm{NRMSE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}\frac{(u_i - v_i)^2}{u_i^2}}, \tag{3}$$

115 and a distortion to the structure function of the precision-trimmed fields, which is defined as follows

$$X(r) = \sqrt{\sum_{i=r}^{N}\frac{(u_i - v_{i+r})^2}{N-r}}, \tag{4}$$

where the integer argument r is called offset.

To illustrate the features of the precision trimming we will use two synthetic arrays: a random process with “-5/3” spectrum over a background (Fig. 2a) and a random Gaussian noise with a background (Fig. 2b). The former is surrogate of a geophysical

https://doi.org/10.5194/gmd-2020-239
Preprint. Discussion started: 30 July 2020

signal with high autocorrelation, whereas, the latter corresponds to a signal with high stochastic component. Each array is of 4096 values long. The exact code used to generate the signals and all the figures of this paper can be found in the Supplementary material.

The summary of NRMSE of the signal in Fig. 2a after trimming precision with the considered methods is summarised in Table 2. The results fully agree with Table. 1: the NRMSE introduced by every method is about half of the maximum error for a single value. Along with above-mentioned methods we have added two ways of rectifying bit-groomed values: *groomhalf*, where we apply *halfshave* to the Bit-Groomed array, and *groomav*, where a simple two-point moving average applied to the data.

As expected, *groomhalf* results in exactly the same values as *halfshave*, and therefore has identical NRMSE. The scores for *halfshave* and *round* are slightly different due to the limited sample size and limited number of discrete levels for the samples. Notable, that for smooth signals like one in Fig. 2a, the moving average of bit-groomed values gives smaller NRMSE than for all other methods. The reason is in the smoothness of the signal that has well-correlated adjacent points, while bit-trimming errors are less correlated.

The structure functions of the signal from Fig. 2a processed with trim-precision are given in Fig. 3. All panels have the structure function of the original signal (“ref” curve) plotted along with curves for the processed signal. Since the structure function is not sensible to the offsets, the plots for *shave*, *halfshave* and *groomhalf* (panels a, c, and e) are identical. Panel d slightly differs form them due to statistical differences mentioned above. The Bit Grooming algorithm produces quite large peaks at odd offsets, whereas the values at even offsets are equal to corresponding values for *shave* or *halfshave*. The running average produces the structure functions that is closer to the original one since moving average partly restores the original smooth signal. The steeper structure function for smaller offsets in Fig 3f is a result of smoothing by the moving average.

The situation is different for the uncorrelated random signal from Fig. 2b. For this signal the reference structure function is flat, and the remaining oscillations are due to the sampling errors. As in the previous case, the increase of the number of keep-bits makes the structure function closer to the reference one, however the offset also depends strongly on the location of the background and the variance with respect to the discretisation levels. As in the case of the smooth signal, Bit Grooming produces oscillating distortions of the structure function, that can be removed by applying *halfshave*. The moving average notably reduces the variance of the original signal due to its the uncorrelated nature.

5 Conclusions

A simple method for trimming precision by rounding a mantissa of floating-point numbers has been implemented and tested. The method has twice smaller quantization error than a widely used Bit Grooming method (Zender, 2016), that has been used by default in NCO.

Bit Grooming, besides being suboptimal, leads to substantial distortion of multipoint statistics of scientific data sets. The “halfshave” procedure can be used to partially recover the precision and remove excess distortions from two-point statistics Bit-Groomed datasets.

Both methods have been implemented in Python. The implementation is available under BSD license.

Code availability. The implementation in C has been merged to the NCO master branch. The source code of NCO is available from http://nco.sourceforge.net (last accessed July 14, 2020) under BSD License. The Python3 code used to generate the example statistics and containing the precision-trimming functions is available from the supplementing material and can be used under BSD License.

Competing interests. The author declares no competing interests

Acknowledgements. The author would like to thank Prof. Mikhail Sofiev, and Dr. Viktoria Sofieva from the Finnish Meteorological Institute for fruitful discussions, and Charles Zender for the NetCDF operators software (NCO) and for being open and quick for verifying and accepting the patches for it.

https://doi.org/10.5194/gmd-2020-239
Preprint. Discussion started: 30 July 2020

References

ANSI/IEEE: IEEE Standard for Binary Floating-Point Arithmetic, ANSI/IEEE Std 754-1985, pp. 1–20, https://doi.org/10.1109/IEEESTD.1985.82928, 1985.

Zender, C. S.: Bit Grooming: statistically accurate precision-preserving quantization with compression, evaluated in the netCDF Operators (NCO, v4.4.8+), Geoscientific Model Development, 9, 3199–3211, https://doi.org/10.5194/gmd-9-3199-2016, 2016.