Error Distributions of Lossy Floating-Point Compressors

Peter Lindstrom[*]

Abstract
With ever increasing volumes of data being generated in scientific simulations, experiments, and observations, storage and bandwidth concerns are mounting. As a means of data reduction, lossless data compression is largely ineffective when applied to such floating-point data, and consequently much recent work has focused on *lossy compression* methods that only approximately reconstruct the data by allowing for small errors. When such approximated data sets are used in data analysis, it is important to understand how errors due to compression are distributed and how they propagate to impact the accuracy of the analysis.

In this paper we perform an empirical study of the statistical distributions of compression-induced errors in scientific data for a number of state-of-the-art data compressors. We find that compression schemes based on scalar quantization tend to give uniformly distributed errors that are weakly data-dependent, and that transform- and decomposition-based methods tend to give Laplace or normally distributed errors. With the exception of the FPZIP compressor, we find the errors to be unbiased with zero mean. We further analyze the error distribution of the ZFP compressor and show using the central limit theorem that it tends to a normal distribution. We conclude with an examination of correlation, both between the function being compressed and its errors and within the error signal itself. Our results suggest that transform-based compression methods more reliably reduce autocorrelation, especially at high compression ratios.

Key Words: Lossy compression, floating-point data, approximation, error distribution, correlation

1. Introduction

With a sustained exponential trend in compute power due to Moore's law and increasing levels of parallelism, large-scale scientific simulations routinely produce petabyte-sized data sets. Similar increases are seen at experimental and observational facilities due to advances in sensing technology. Not only do such large data sets rarely fit on fast persistent storage such as spinning disk, but as simulations are increasingly run remotely due to contention for scarce compute resources, data sets must frequently be transferred from the supercomputer or experimental facility where they were generated to local storage. The time needed to transfer a petabyte of data over the internet can be substantial.

One approach to mitigating this storage and bandwidth problem is to apply *data compression* to reduce the data size. One often distinguishes between two forms of compression: *lossless compression*, which ensures that each and every bit of data is faithfully preserved, and *lossy compression*, where the numerical data is approximated, often to within a user-specified error tolerance.

It is well-known that general-purpose lossless compressors like GZIP allow for only modest reductions of floating-point data; usually on the order of a few percent reduction. This is perhaps not surprising since such compressors are designed to exploit repeated patterns that are often present in text documents, but which rarely appear in high-precision numerical data. Even the best lossless compressors designed specifically for floating-point data [1, 6, 13, 24] are able to reduce double-precision floating-point data only on the order of 10% (see Fig. 1). This is due to the inherent randomness of the majority of mantissa bits in a floating-point number, which for all practical intents exhibit no pattern and cannot be compressed.

[*]Lawrence Livermore National Laboratory, 7000 East Avenue, L-478, Livermore, CA 94550.

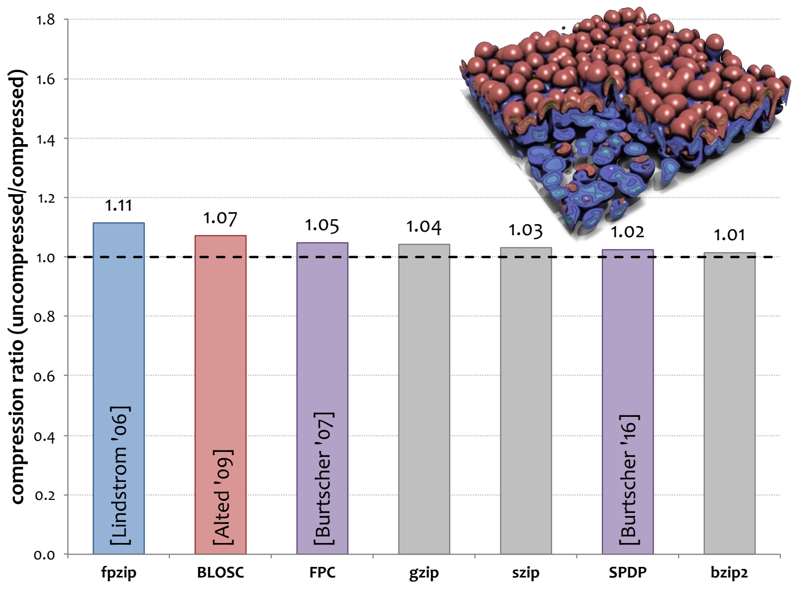


Figure 1: The results of applying some of the best floating-point compressors to a fairly smooth double-precision 3D viscocity field from a turbulence simulation. Even the most effective compressor, FPZIP, reduces the data in size by only 11%.

As a result, most recent data compression work in the high-performance computing community has focused on *lossy compression* [2, 5, 7, 14, 16, 17, 19, 21, 28, 30, 33]. In lossy compression the numerical data is not exactly recovered, but is approximated. Usually this is acceptable since the data itself tends to involve many error terms. For example, experimental and observational data is subject to finite precision measurements and intrinsic measurement noise. Numerical simulation inevitably involves round-off, truncation, iteration, and model errors, as well as initial conditions of limited accuracy. Hence it is often acceptable to approximate a fairly large number of the floating-point bits, especially for data analysis and visualization. The tradeoff of such an approximation is substantially improved compression ratios. For instance, for visualization applications it is not uncommon to see lossy compression ratios on the order of 100:1 with little or no loss in visual quality.

In statistical analysis, it is common to assume that the error (e.g. due to lossy compression and other sources) is i.i.d., which simplifies the analysis. For this assumption to hold, the error, δ, in the function value, f, must be independent of f and exhibit no autocorrelation. In many cases, it is further preferable that the error conforms to Gaussian white noise [31]. In other words, δ is normally distributed and has a uniform power spectrum. The normality assumption helps in the analysis of error propagation, whereby normal errors remain normal under addition and subtraction, e.g. in finite difference derivative estimates.

The issue of the distribution and dependence of compression-induced errors in floating-point data has received relatively little attention. Baker et al. [3, 4] present metrics for assessing the impact of compression errors on climate analysis. Wegener [34] describes the APAX Profiler, a graphical tool that reports various error metrics for compressed data. A more recent effort along these lines is the work by Tao et al. [31], who describe the Z-CHECKER tool for error analysis. These papers focus primarily on the analyses supported by these tools.

In this paper we perform an empirical study of ten lossy compressors and examine their error distributions, correlations between fields and errors, and autocorrelation of errors. We evaluate the error of each compressor in terms of its **distribution**, **bias**, **correlation** with the function itself, and **autocorrelation**.

We begin with some background material on compression and statistical measures, followed by an examination of error distributions. As we have a particular interest in the ZFP compressor, we provide an explanation for why it yields approximately normally distributed errors. We continue by examining the relationship between the function and its errors, followed by a section on autocorrelation of errors. We conclude with a discussion of the impact of errors in statistical analysis.

2. Preliminaries

Let $f : \mathbb{R}^d \to \mathbb{R}$ be a scalar function over a d-dimensional domain. In this paper, we consider the case $d = 3$ and assume that f has been discretized onto a uniform Cartesian grid. We will use "function," "field," and "data" synonymously to refer to f. Let $\tilde{f}$ be the corresponding approximate field resulting from lossy compression and decompression. The (signed) *error* field is given by $\delta = \tilde{f} - f$.

Most lossy compressors provide parameters to balance the tradeoff between error and compressed size. Examples include tolerances on either the *absolute error*

$$\delta_{abs}(\mathbf{x}) = \tilde{f}(\mathbf{x}) - f(\mathbf{x}), \tag{1}$$

or *relative error*

$$\delta_{rel}(\mathbf{x}) = \frac{\tilde{f}(\mathbf{x}) - f(\mathbf{x})}{|f(\mathbf{x})|}. \tag{2}$$

Other examples include bounds on the compressed size, or indirect size control by specifying, for instance, the number of codewords to use in scalar or vector quantization or a threshold on wavelet coefficients.

The bit *rate*, r, is a measure of amortized storage size in bits per scalar value. Expressing the compressed size this way allows for straightforward comparison with uncompressed data, e.g., $r = 32$ for single precision and $r = 64$ for double precision uncompressed floating-point values.

The *covariance* between f and g is given by

$$\mathrm{cov}(f, g) = E[(f - E[f])(g - E[g])] \tag{3}$$

and the *correlation* by

$$\mathrm{corr}(f, g) = \frac{\mathrm{cov}(f, g)}{\sigma_f \sigma_g}, \tag{4}$$

where E denotes expectation and σ^2 denotes variance.

The *autocorrelation* function, $R(f)$, is given by the correlation of f with translations of itself. In other words, R is given by the convolution

$$R(f) = \hat{f} \star \hat{f}', \tag{5}$$

where

$$\hat{f} = \frac{f - E[f]}{\sigma_f} \tag{6}$$

has zero mean and unit variance, and where $\hat{f}'(\mathbf{x}) = \hat{f}(-\mathbf{x})$. We define the *autocorrelation coefficient* as $\|R\| = \sqrt{E[R^2]}$.

Traditionally autocorrelation is given for functions of one variable, as is also assumed by Z-CHECKER [31], which flattens multiple dimensions and reports only the autocorrelation as a function of a delay in one dimension. However, in multiple dimensions, d, correlations along each of the d dimensions are possible, and it is generally more informative to view autocorrelation as a multidimensional function.

A naive implementation of the autocorrelation function can be prohibitively expensive, as it involves joint self-convolutions along multiple dimensions. To evaluate this function, we take advantage of the Fourier convolution theorem, which allows us to reduce convolutions in the time domain to multiplications in the Fourier domain. We first shift the function and normalize it so that it has zero mean and unit variance. We then compute the discrete Fourier transform, compute the square magnitude (this multiplication takes the place of convolution in the time domain), and then apply the inverse Fourier transform, resulting in the autocorrelation function $R(x, y, z)$.

The Fourier transform assumes periodic domains; an assumption that generally does not hold for scientific data. Hence, as is common, we pad the sampled function with zeros along each dimension in the autocorrelation computation, doubling the size of the domain in each dimension. Once R has been computed, this padding is removed.

3. Compressors

In this section we provide a brief summary of each of the compressors used in our study. We focus in particular on how the compressors approximate the data with the goal of characterizing the error.

3.1 FPZIP

FPZIP [24] is a predictive compressor for scalar fields defined on d-dimensional grids. FPZIP uses the *Lorenzo predictor* [15] to predict the next "corner" value of a hypercube from its previously encoded $2^d - 1$ hypercube neighbors. This predictor is exact [20] for fields that satisfy $\frac{\partial^d f}{\partial x_1 \partial x_2 \cdots \partial x_d} = 0$. FPZIP reduces data through efficient entropy coding of the residuals between predictions and actual values.

FPZIP was originally designed for *lossless* compression. A lossy extension is based on the idea of first *truncating* floating-point values by zeroing some number of least significant mantissa bits, effectively rounding values toward zero. The remaining, truncated floating-point values are then compressed losslessly.

FPZIP has gained popularity within the climate science community, where it has proven to be resilient to the types of analyses commonly used in this science domain [4]. As discussed in [3], however, FPZIP's systematic rounding toward zero sometimes leads to energy loss that may introduce bias in the analysis. This biasing of errors is also examined and highlighted in this paper. We used FPZIP version 1.1.0 [22] in our study.

3.2 LZ4A, LZ4P

Kunkel et al. [17] propose two simple approaches to lossy compression that nevertheless sometimes prove quite effective. The first one, *abstol* (here called LZ4A), performs uniform scalar quantization followed by general purpose LZ4 compression, which ensures a bounded absolute error. The other, *sigbits* (here called LZ4P), truncates floating-point values by discarding least significant mantissa bits, thus bounding the relative error and ensuring fixed precision. As with LZ4A, LZ4 compression is then applied. We re-implemented these two schemes for our study. We slightly modified LZ4P by replacing the discarded mantissa bits with $100\ldots0$ (i.e. we used a mid-riser quantizer) rather than simply zeroing them, as is done in FPZIP, which as we shall see avoids FPZIP's biased errors.

3.3 SQ

The SQ adaptive scalar quantization compressor [16] reduces a field of n floating-point values to a discrete set of m disjoint intervals, with $m \ll n$. Given an absolute error tolerance, ϵ, SQ first sorts the data and then greedily grows sets S_i of monotonically increasing values such that $\max S_i - \min S_i \leq \epsilon$. Each such set is represented by a *prototype* value, given by $\hat{s}_i = \operatorname{mean} S_i$, which ensures minimal root-mean-square error. Each value $f \in S_i$ is then represented by the set index i and is reconstructed as $\hat{s}_i$, where $-\epsilon \leq \hat{s}_i - f \leq +\epsilon$. Note that $\hat{s}_i$ can get arbitrarily close to either $\min S_i$ or $\max S_i$ depending on the distribution of values in S_i. In practice, $\hat{s}_i$ is near the middle of the range of S_i such that the error is often within $\pm\frac{\epsilon}{2}$.

SQ stores the *codebook* of m scalar prototypes $\{\hat{s}_i\}_{i=1}^{m}$ and n set indices, i, and then applies LZMA general purpose lossless compression to the codebook and indices to further reduce data size. Due to its simplicity, we wrote our own implementation of SQ.

3.4 HVQ

The hierarchical vector quantizer by Schneider and Westermann [28] generates a two-level hierarchy of blocks. The mean of each block of $4 \times 4 \times 4$ values is subtracted from the fine level and then used to construct the coarse level. Fine-level blocks are treated as vectors of $4^3 = 64$ values; coarse-level vectors are formed as $2^3 = 8$ mean values. Two independent codebooks with the same number of vector prototypes are then generated using an optimization strategy. Each 2^3 and 4^3 block of values is quantized to the closest prototype vector. We obtained the authors' own implementation of HVQ [27].

3.5 SZ

The SZ compressor is based on polynomial prediction. Whereas the original algorithm [11] uses constant, linear, and quadratic one-dimensional prediction, the more recent version [12, 30] used in this study employs multidimensional prediction that generalizes Lorenzo prediction [15]. Given a user-specified absolute error tolerance, ϵ, SZ defines an interval of size 2ϵ centered on the prediction, as well as $2^{m-1} - 1$ similarly-sized non-overlapping intervals in either direction (m is a parameter that can be chosen by the user). If the actual value falls in one of these intervals, an m-bit index is sufficient to identify it, and the value is approximated by the interval midpoint with absolute error at most $\pm\epsilon$. One m-bit code is reserved to flag values that do not fall in one of the intervals, and such "mispredictions" are explicitly corrected using a separate coding scheme. The resulting codestream is then further compressed using a combination of Huffman coding and GZIP.

In effect, SZ employs uniform scalar quantization of prediction *residuals* rather than original values, with a two-level coding scheme that employs m bits to encode small (in magnitude) residuals and longer codes for large residuals. We used SZ version 1.4.9 [12] in our study.

3.6 ISABELA

The ISABELA compressor [19] was primarily designed to compress data that, untransformed, is challenging to compress, such as noisy signals that exhibit very little autocorrelation (i.e. smoothness). To make uncorrelated data more correlated, ISABELA first sorts the values within a small window of consecutive values. The resulting, much smoother signal is then approximated using a few control points of a spline, and residuals to this fit larger than a tolerance are corrected so as to bound the relative error. Although this spline is often a good fit that results in small residuals, there is a substantial cost associated with encoding the permutation of sorted values. In practice, ISABELA is competitive only when compressing functions that are virtually uncorrelated. For our study, we used version 0.2.1 [18].

3.7 VAPOR

VAPOR [9] is a visualization and data analysis tool developed at NCAR for climate and weather data. VAPOR supports wavelet-based compression using the popular CDF 9/7 biorthogonal wavelets [10]. Lossy compression is achieved by discarding some number of smallest wavelet coefficients. VAPOR stores a *significance map* that encodes the locations of remaining wavelet coefficients. This is a rather basic form of wavelet compression, and more sophisticated coders like SPIHT [26], SPECK [25], and EBCOT [32] have been developed for image compression. Surprisingly these more advanced wavelet coding techniques have seen little use in the scientific computing community. For our study, we used VAPOR version 2.4.2 [8].

3.8 TUCKER

The wavelet and discrete cosine transform (DCT) attempt to decorrelate a function using bases in which the function is expected to be sparse. A sparse representation implies few nonzero coefficients to encode, or many small coefficients that can be zeroed with little impact on accuracy. For 2D data, the basis in which the function is sparsest is given by the singular value decomposition (SVD). With maximal sparsity comes a cost, however: the basis vectors are data-dependent and must be encoded explicitly, which introduces a high overhead in relation to wavelets and DCT, for which the basis functions are already known by the decoder. The Tucker tensor decomposition is a generalization of SVD to three dimensions. Here the number of transform coefficients greatly outweighs the storage needed for basis vectors, and hence compression based on tensor decomposition becomes an attractive choice for $d \geq 3$.

Although several tensor-based compression techniques exist, e.g. [2, 5, 29], we have chosen the one proposed by Ballester-Ripoll and Pajarola [5]. This method uses rank reduction and non-uniform coefficient quantization coupled with Huffman based significance coding. We obtained the authors' own Matlab implementation of TUCKER.

3.9 ZFP

The ZFP compressor [21] was originally designed to be a compressed array primitive that supports random access, but can be used also for streaming (sequential) compression. ZFP, similar to HVQ, partitions d-dimensional arrays into blocks of 4^d values. Each such block is compressed independently using (1) a block-floating-point representation with a single common exponent per block, (2) a decorrelating transform similar to the discrete cosine transform, followed by (3) embedded encoding that bears some resemblance to SPIHT [26]. The embedded encoder transmits transform coefficients one "bit plane" at a time, from most to least significant bit. As a consequence, the compressed bit stream for a block can be truncated anywhere, for instance to ensure a fixed rate, which enables random access at block granularity. By truncating at other points, ZFP also supports fixed accuracy (bounded absolute error) and fixed precision via variable-rate blocks. We used ZFP version 0.5.1 [23] in fixed-accuracy mode in our study.

Application	*Physics*	*Grid*	*Precision*
MIRANDA	turbulence	$384 \times 384 \times 256$	double
S3D	combustion	$256 \times 384 \times 384$	double
QMCPACK	quantum mechanics	$69 \times 69 \times 115$	float
CESM	climate	$288 \times 192 \times 28$	double
ISABEL	weather	$500 \times 500 \times 100$	float

Table 1: Floating-point data sets used in our study.

4. Experimental Setup

We used several data sources in our experiments that span a variety of application domains, as outlined in Table 1. All data sets are defined on Cartesian grids and consist of multiple floating-point fields, out of which we chose a representative subset. Several of our experiments used the MIRANDA viscocity field (Fig. 1), which is signed and highly peaked around zero, and thus spans many floating-point exponents that are well preserved by the compressors that bound relative error.

To examine error distributions, we set a fixed absolute error tolerance and ensured that each compressor met this tolerance. For some compressors, such as LZ4A, SQ, SZ, and ZFP, this is achieved trivially by specifying the error tolerance as the key compression parameter. For compressors that bound only the relative error, such as FPZIP, LZ4P, and ISABELA, as well as compressors that provide only indirect error control, such as HVQ, VAPOR, and TUCKER, we tested a range of compression parameters, starting at a low rate and progressively increasing the rate until the error tolerance was met, e.g. by successively doubling the codebook size in HVQ. We used a similar approach to producing rate plots, e.g. by successively doubling the error tolerance.

5. Error Distributions

We begin by examining absolute error distributions for the various compressors. Fig. 2 shows for each compressor six different error distributions that correspond function values that lie in three positive and three negative ranges of f. The purpose of showing several distributions rather than just one is to ascertain whether the error, $\delta = \tilde{f} - f$, is dependent on the magnitude of f, as we would expect for compressors that bound the relative error (i.e. we should then see smaller errors and narrower distributions for smaller function values). Conversely, we expect the distribution to be largely independent of f for the other compressors.

As is evident in Fig. 2, the three compressors that bound relative errors—FPZIP, LZ4P, and ISABELA—show tighter absolute errors over ranges of smaller (in magnitude) function values. A striking result is the bias in errors resulting from FPZIP, as previously conjectured, with error distributions not being zero centered. This is a consequence of always truncating values, rounding them toward zero, which results in negative errors for positive values, i.e. $0 \leq \tilde{f} \leq f$ such that $\delta = \tilde{f} - f \leq 0$, and vice versa. Our implementation of LZ4P rounds to nearest rather than toward zero, and therefore gives similar but unbiased distributions. It is also evident that these rounding errors are uniform, which supports the conjecture that the lost least significant mantissa bis are essentially uniformly random.

The ISABELA compressor also bounds relative errors, but compresses residuals resulting from least-squares polynomial fits. These residuals tend to be more peaked around zero (as one might expect from a good predictor), and follow distributions more similar to Laplacian or Gaussian.

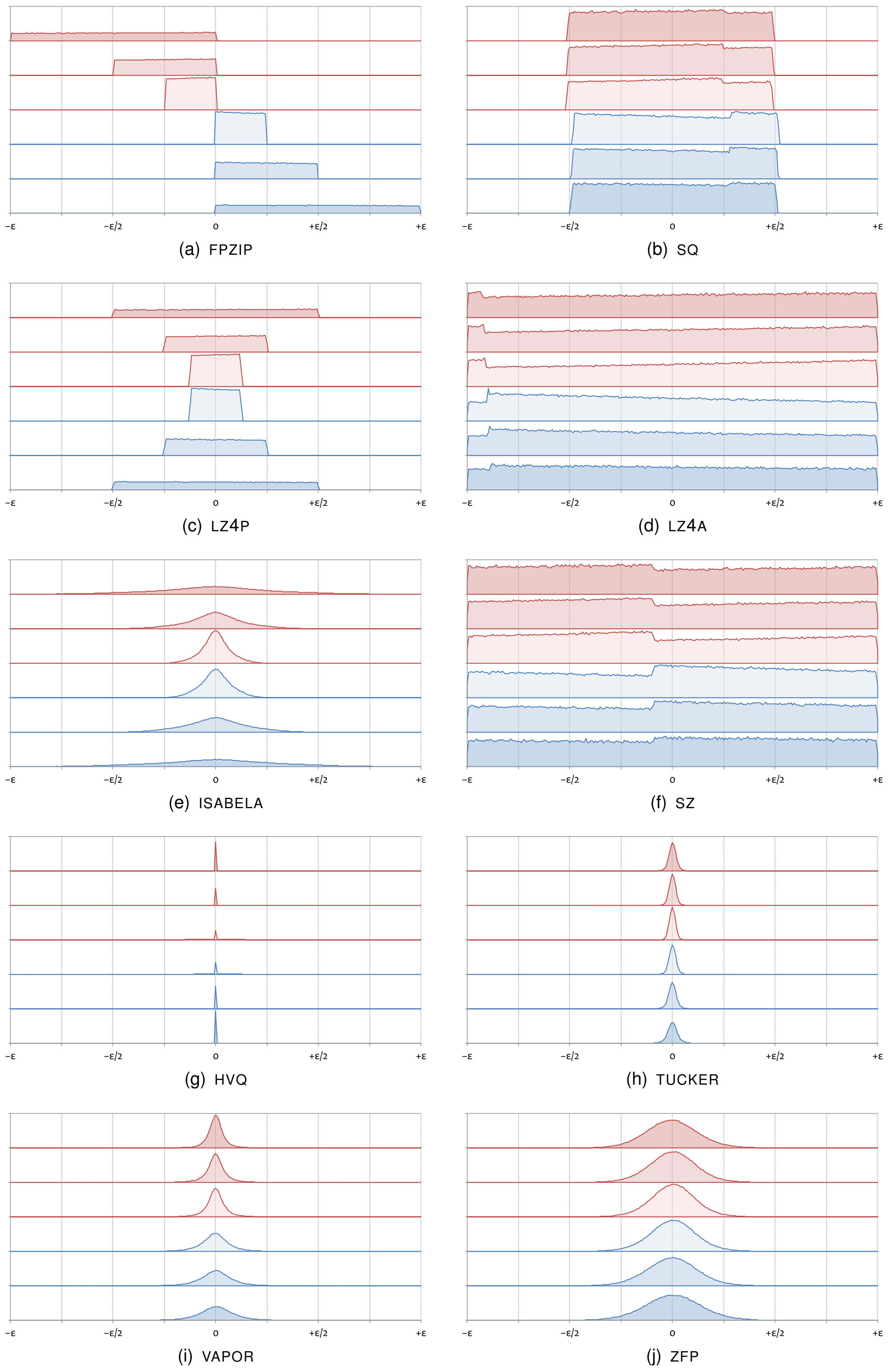


Figure 2: Compression error distributions for the field shown in Fig. 1. Each plot shows six distributions for six different ranges of function values, f: From top to bottom, $[2, 4)$, $[1, 2)$, $[\frac{1}{2}, 1)$, $(-1, -\frac{1}{2}]$, $(-2, -1]$, $(-4, -2]$. Compression settings were chosen to ensure that $\max |\delta_{abs}| < \epsilon = \frac{1}{8}$. FPZIP, LZ4P, and ISABELA bound the relative error, as is evident from the error distribution dependence on f.

The uniform scalar quantizers—SQ, LZ4A, and SZ—not surprisingly tend to exhibit uniformly distributed errors. As is evident, the distribution of the original data has some data-dependent influence on these distributions, with clear patterns emerging. SQ conservatively assumes that the prototype could be either at the top or bottom of the range it is assigned to, yet almost always is in the middle, which explains why its range of errors is in practice narrower than for LZ4A and SZ. These latter two compressors both accept any approximation within the tolerance, and therefore span the full $[-\delta, +\delta]$ error range. Some weak dependence on distribution of f can be observed, which is difficult to undo for these compressors.

The HVQ plot exhibits a distribution quite different from those generated by the other compressors. In fact, when $2 \leq |f| < 4$, some 90% of errors are within single precision machine epsilon (the HVQ implementation converts the double precision input data to single precision), and the HVQ error distribution resembles a sharp spike at zero. Zooming in on the plot by a factor of 1,000,000 reveals a distribution similar in shape to Fig. 2(c). That is, as f is halved, the floating-point resolution is doubled, resulting in smaller, uniformly distributed errors.

HVQ required 2^{16} bins to stay within the error tolerance, which is one ninth the number of 4^3-sized blocks in this data set. In other words, a substantial fraction of blocks were represented nearly losslessly, while the difficulty of generating such a large codebook still causes occasional large errors, making HVQ unsuitable as a bounded-error compressor.

Vector quantization (VQ) can be fairly effective and efficient for low-precision data, and VQ remains a popular choice for visualization applications due to its simplicity and speed of decompression. However, for applications that demand high precision and accuracy, the size codebooks needed and the difficulty of generating them make VQ computationally impractical for precisions beyond 16 bits.

The transform-based methods, VAPOR, TUCKER, and ZFP, produce error distributions that appear (close to) normal. An explanation for this phenomenon is discussed below. We note that ZFP's distribution is far wider than those of VAPOR and TUCKER, while still not spanning the full range $[-\epsilon, \epsilon]$. In order to bound the absolute error, ZFP must conservatively assume a worst-case (rapidly oscillating) input that in principle could reach the full error tolerance, though in practice errors span only a fraction of the interval. The situation for VAPOR and TUCKER is different in that these methods do not provide theoretical guarantees on the error. Providing tight error bounds for these two compressors is made difficult by the complex interactions of many overlapping basis functions.

5.1 Normality of the ZFP Error Distribution

In this section we analyze the error distribution of the ZFP 0.5.x compressor. Although we do not provide a rigorous proof, the purpose of this section is to give some intuition for why ZFP compression errors tend toward normal.

Although the basic components of the algorithm have remained the same, ZFP has evolved since the original publication [21]. The algorithm associated with the version used in this paper is summarized in the software documentation [23]. From the standpoint of error distribution, the two salient points are ZFP's *negabinary* representation of signed transform coefficients and its inverse decorrelating transform. This separable linear transform, which is applied once along x, y, and z, is given in matrix form by:

$$A^{-1} = \frac{1}{4}\begin{pmatrix} 4 & 6 & -4 & -1 \\ 4 & 2 & 4 & 5 \\ 4 & -2 & 4 & -5 \\ 4 & -6 & -4 & 1 \end{pmatrix} \tag{7}$$

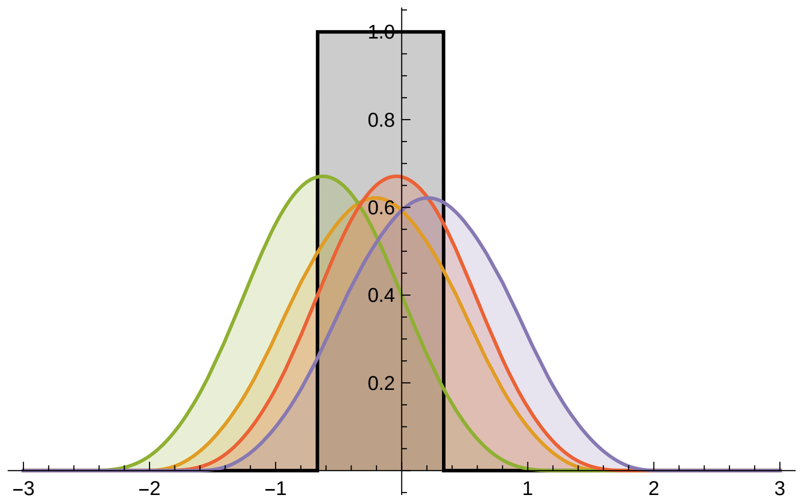


Figure 3: The uniform distribution $\mathcal{U}(-\frac{2}{3}, \frac{1}{3})$ (in black) and the four Gaussian shaped distributions resulting from applying ZFP's inverse transform (Eq. (7)) once to four uniformly distributed coefficients.

By truncating (zeroing) transform coefficient bits smaller in place value than some binary quantization threshold, 2^q, the resulting normalized error $\hat{\delta} = 2^{-q}\delta$ in each transform coefficient lies either in $(-\frac{2}{3}, \frac{1}{3})$ or in $(-\frac{1}{3}, \frac{2}{3})$, depending on whether q is odd or even.[1] Regardless of how the coefficients themselves are distributed, we make the assumption that the discarded bits trail a leading one bit, and that their distribution changes slowly enough that those discarded bits can in practice be treated as uniformly random. As found in [24], this tends to be true of Laplace distributed prediction residuals. Consequently, we may treat the error in coefficients as uniform over the above intervals. We further assume that transform coefficients are not correlated. Indeed, this is the primary purpose of a decorrelating transform, and hence this is a reasonable assumption.

As a consequence of our assumptions, we seek to reconstruct a function from i.i.d. coefficients via repeated application (once per dimension) of the inverse transform A^{-1} above. The result of this linear transform is a weighted sum of four uniformly distributed random variables. As an aside, when this sum is not weighted, the resulting distributions are given by convolutions of hat functions with themselves, which gives rise to B-spline basis functions of increasing order. After one application, we obtain piecewise cubic B-splines. In three dimensions, we eventually obtain degree-9 B-splines. In the limit, B-splines converge to Gaussian functions. Moreover, even if the errors in our coefficients were not uniformly distributed, repeated convolutions will by the central limit theorem cause the transformed coefficients to approach normal.

Returning to the case of weighted combinations, the distributions resulting from application of A^{-1} are also piecewise cubic, and very close to normal. These four distributions as well as the uniform distribution $\mathcal{U}(-\frac{2}{3}, \frac{1}{3})$ are shown in Fig. 3. Clearly one application of A^{-1} brings the uniformly distributed errors closer to normal. In 2D, we apply A^{-1} to four identical copies of one of those four distributions; while in 3D we apply A^{-1} yet one more time. This repeated "smoothing" of error distributions gives rise to the near-normal distributions shown in Fig. 2(j).

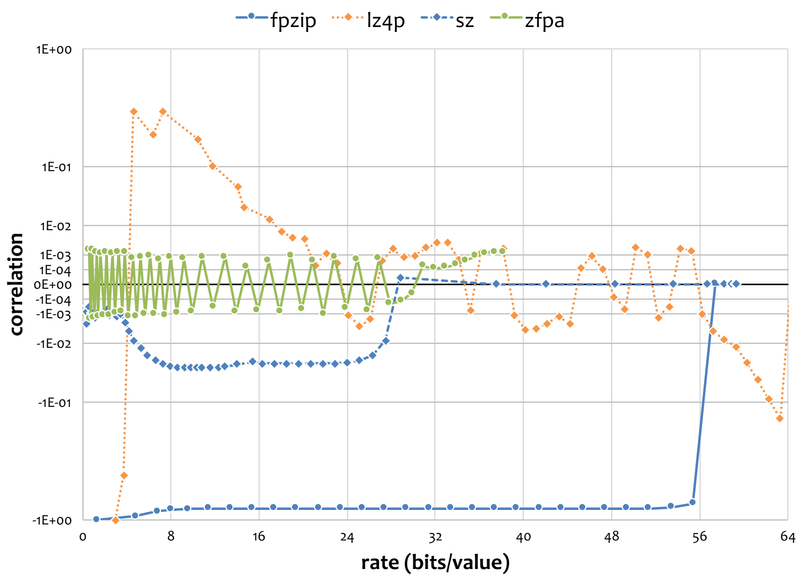


Figure 4: Correlation between the function and compression-induced errors in the function. Note the non-linearity of the vertical axis. Because FPZIP truncates values, the errors are strongly (negatively) correlated with the values themselves. Zero correlation is achieved only once lossless compression is attained, above 56 bits/value. LZ4P achieves decorrelated errors once both exponent and mantissa bits are retained, while SZ and ZFP both exhibit near zero correlation.

6. Correlation of Function with Error

As stated earlier, one desirable characteristic of a lossy compressor is that its errors, δ, are independently distributed. One way of testing this property is to examine whether δ is correlated with the uncompressed function, f. We note that compressors like FPZIP, which truncates values and rounds them toward zero, will always yield nonnegative errors for negative function values and vice versa. Thus the sign of the error is inversely correlated with the sign of the function, which implies at least weak correlation between FPZIP errors and the data.

Fig. 4 plots the correlation $\mathrm{corr}(f, \delta)$ as a function of rate for the MIRANDA viscocity field. As just conjectured, FPZIP exhibits not only a weak anticorrelation between function and error but a very strong one, with the Pearson correlation coefficient near negative one. We see that LZ4P, which rounds to nearest rather than to zero, avoids this bias in sign and yields far lower correlation. Although obvious in hindsight, the design of FPZIP would have benefitted from such improved rounding.

We further see from this plot that the errors for both ZFP and SZ are nearly fully independent (note the nonlinear scale of the vertical axis). We conjecture that the oscillating pattern shown by ZFP is due to its use of a negabinary representation, where each subsequent point on the curve corresponds to alternatingly adding a bit plane of positively or negatively valued bits.

7. Autocorrelation of Errors

Another desirable characteristic is the absence of autocorrelation in the error field. Whereas Z-CHECKER provides correlograms by treating the multidimensional field as a linear time

[1]This is due to the alternating sign of bits in the negabinary representation.

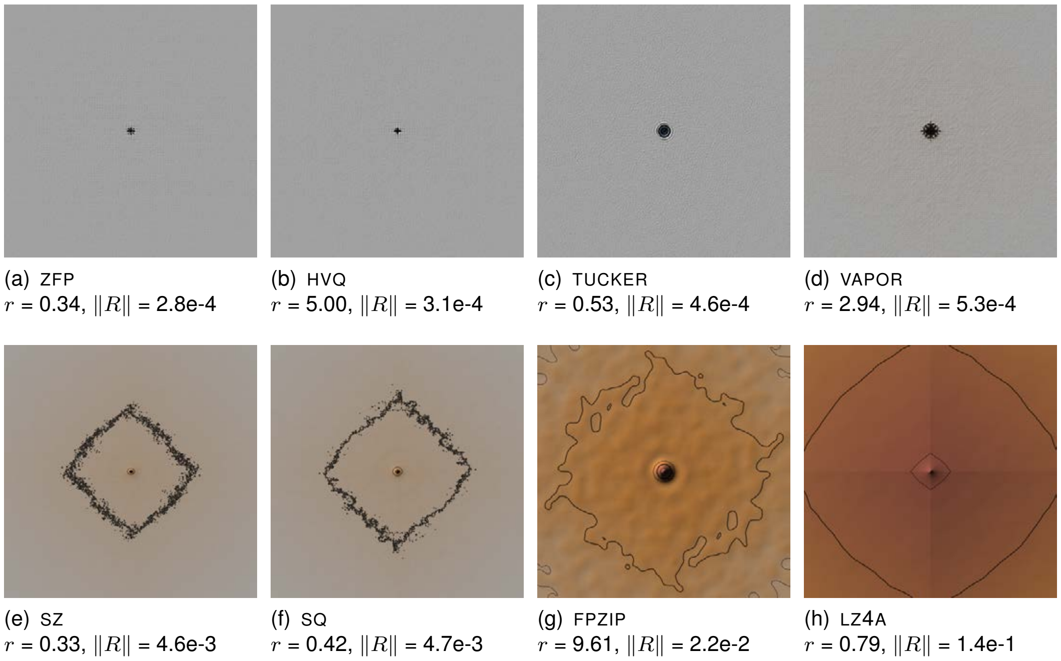


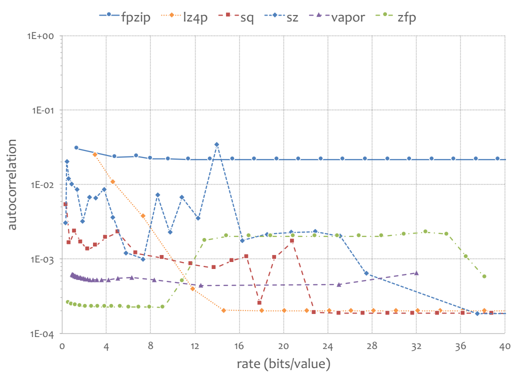
Figure 5: Autocorrelation plots for errors in the field shown in Fig. 1 and distributions shown in Fig. 2. Only a 2D xy slice of the 3D autocorrelation function $R(x, y, z)$ is here shown, with a zero lag in the z direction. Gray indicates no correlation while saturated red indicates maximal positive correlation. An ideal response is a Dirac delta, with $R(0, 0, 0) = 1$ and $R(x, y, z) = 0$ elsewhere. r represents the rate in bits/value (lower is better); $\|R\|$ is the root mean square autocorrelation value over the 3D domain (lower is better).

series, we note that correlations may occur along any and all dimensions. Indeed, the autocorrelation function for a multidimensional field is most naturally expressed as another field of the same dimensions.

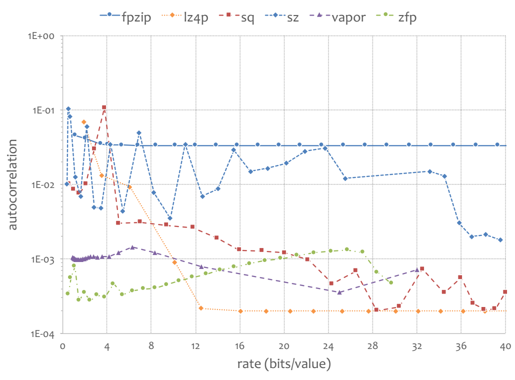
Fig. 5 shows 2D slices of the 3D autocorrelation function, $R(\delta)$, of the compression error, δ. Since any function is perfectly correlated with itself, while we desire a lack of correlation between error values and their neighbors, the “ideal” R is a Dirac delta, with $R(0, 0, 0) = 1$ and $R(\mathbf{x}) = 0$ elsewhere. In these visualized slices of $R(x, y, 0)$, which also show contour lines, we would thus like to see a single spike in the middle of a uniform gray background, representing zero correlation.

The figure shows the autocorrelation for the error distributions visualized in Fig. 3. Because $R(\mathbf{x}) = R(-\mathbf{x})$, the slices exhibit symmetry. The figure also lists the rate and autocorrelation coefficient $\|R\|$ (lower is better for both). As is evident, the top row of this figure shows autocorrelation functions that approximate Dirac deltas, while the compressors in the bottom row do not fare as well. SZ, SQ, and—to a larger extent—LZ4A show diamond-shaped contours, indicating substantial correlation even far away from the center. ZFP gives both a low rate and autocorrelation, making it the preferred compressor in this example.

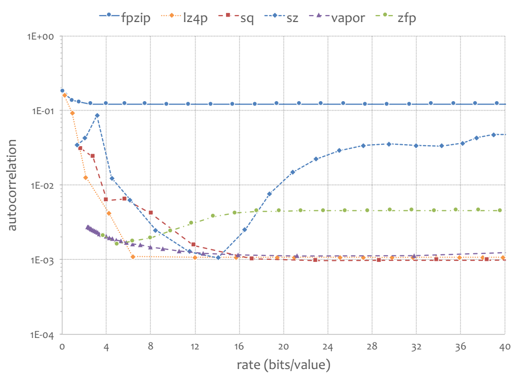
Fig. 6 plots the autocorrelation coefficient as a function of rate for four different fields. Although the plots differ somewhat from one field to another, we observe some consistent trends. First, FPZIP, essentially retains the autocorrelation present in f, and gives a consistently high $\|R\|$. Interestingly, simply changing FPZIP’s rounding rule from downward to nearest, as in LZ4P, reduces autocorrelation by two orders of magnitude. However, this reduction occurs only for high enough rates, when a large enough number of mantissa bits are kept.



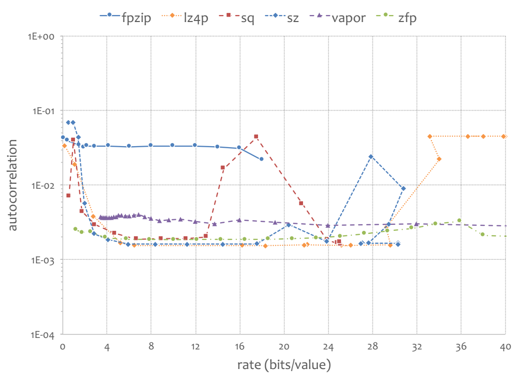
(a) MIRANDA viscocity ($\|R\|$ = 0.030)



(b) S3D oxygen fraction ($\|R\|$ = 0.047)



(c) CESM specific humidity ($\|R\|$ = 0.181)



(d) QMCPACK atomic orbital ($\|R\|$ = 0.044)

Figure 6: Autocorrelation in compression error as a function of rate for various fields. The captions list the autocorrelation coefficient of the function being compressed.

The two methods based on scalar quantization, SQ and SZ, give somewhat erratic and unpredictable autocorrelation curves, where there seems to be no clear pattern between $\|R\|$ and the rate. As with LZ4P, the autocorrelation is highest at low rates. Fig. 6(d) shows how the rate surprisingly changes nonmonotonically for LZ4P and SZ with increasing error tolerances, in addition to a reversal of the trend of decreasing autocorrelation. The two transform-based methods, VAPOR and ZFP, on the other hand yield fairly stable autocorrelation curves that are either flat or slowly varying. Moreover, these two compressors differ in that they give low autocorrelation also at low rates, where arguably compression is of most utility. These results seem to partially contradict the findings from [31], although the authors there report only one-dimensional autocorrelation and for a single rate. Finally, we note that due to $R(0) = 1$, $\|R\|$ is bounded from below by $n^{-1/2}$, where n is the number of grid points, effectively setting a floor on autocorrelation.

8. Conclusions

We have examined the error distributions of a number of state-of-the-art lossy compressors for floating-point data defined on structured grids. Our study reveals that error distributions in practice falls in one of two camps: scalar quantizers give approximately uniform distributions, while transform-based methods give distributions that are nearly normal. We provided an explanation of this normality for the ZFP compressor. We further analyzed the dependence of error distributions on the function being compressed, as well as within the error field itself in terms of autocorrelation.

As found also in [31], the extent to which compression errors show autocorrelation varies by compressor from one data set to another. However, we observe a few general trends. Due to systematic rounding, FPZIP tends to give predictable and consistently high autocorrelation, which can partially be remedied by rounding to nearest quantized floating-point value, as in LZ4P. ZFP and VAPOR exhibit less fluctuations in autocorrelation than scalar quantizers like SQ and SZ as the rate is varied, while ZFP tends to give the lowest autocorrelation among the compressors at low rates of a few bits/value.

This paper is concerned with statistical measures of compression-induced errors. Other considerations not addressed here include the spectral properties of errors and the effect compression has on spectral analysis, as examined further in [21, 31, 34]; the impact of errors on visualization; as well as the impact that errors have on derived fields resulting, for instance, from differential operators such as gradient, divergence, curl, Laplacian, etc. Another area of concern is the impact that compression errors have on the physicality and consistency of analysis, such as the potential for violating conservation laws or physical relationships between fields. We leave these as interesting topics for future work.

Acknowledgments

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344. We wish to thank Franck Cappello for raising the issue of error distributions and Dorit Hammerling for insightful discussions and for encouraging the publication of this work. Finally, we thank the authors of the compressors used in our study for sharing their implementations, and the simulation code leads for making their data available.

References

[1] F. Alted. Why modern CPUs are starving and what can be done about it. *Computing in Science & Engineering*, 12(2):68–71, 2010.

[2] W. Austin, G. Ballard, and T. G. Kolda. Parallel tensor compression for large-scale scientific data. In *IEEE International Parallel and Distributed Processing Symposium*, pages 912–922, 2016.

[3] A. H. Baker, D. M. Hammerling, S. A. Mickelson, H. Xu, M. B. Stolpe, P. Naveau, B. Sanderson, I. Ebert-Uphoff, S. Samarasinghe, F. D. Simone, F. Carbone, C. N. Gencarelli, J. M. Dennis, J. E. Kay, and P. Lindstrom. Evaluating lossy data compression on climate simulation data within a large ensemble. *Geoscientific Model Development*, 9(12):4381–4403, 2016.

[4] A. H. Baker, H. Xu, J. M. Dennis, M. N. Levy, D. Nychka, S. A. Mickelson, J. Edwards, M. Vertenstein, and A. Wegener. A methodology for evaluating the impact of data compression on climate simulation data. In *ACM Symposium on High-Performance Parallel and Distributed Computing*, pages 203–214, 2014.

[5] R. Ballester-Ripoll and R. Pajarola. Lossy volume compression using Tucker truncation and thresholding. *The Visual Computer*, 32(11):1433–1446, 2016.

[6] M. Burtscher and P. Ratanaworabhan. High throughput compression of double-precision floating-point data. In *Data Compression Conference*, pages 293–302, 2007.

[7] Z. Chen, S. W. Son, W. Hendrix, A. Agrawal, W.-k. Liao, and A. Choudhary. NUMARCK: Machine learning algorithm for resiliency and checkpointing. In *International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 733–744, 2014.

[8] J. Clyne. VAPOR version 2.4.2, May 2015. `http://www.vapor.ucar.edu`.

[9] J. Clyne, P. Mininni, A. Norton, and M. Rast. Interactive desktop analysis of high resolution simulations: Application to turbulent plume dynamics and current sheet formation. *New Journal of Physics*, 9(8):301, 2007.

[10] A. Cohen, I. Daubechies, and J.-C. Feauveau. Biorthogonal bases of compactly supported wavelets. *Communications on Pure and Applied Mathematics*, 45(5):485–560, 1992.

[11] S. Di and F. Cappello. Fast error-bounded lossy HPC data compression with SZ. In *IEEE International Parallel and Distributed Processing Symposium*, pages 730–739, 2016.

[12] S. Di and D. Tao. SZ version 1.4.9, Feb. 2017. `https://github.com/disheng222/SZ`.

[13] N. Fout and K.-L. Ma. An adaptive prediction-based approach to lossless compression of floating-point volume data. *IEEE Transactions on Visualization and Computer Graphics*, 18(12):2295–2304, 2012.

[14] E. Gobbetti, J. A. I. Guitán, and F. Marton. COVRA: A compression-domain output-sensitive volume rendering architecture based on a sparse representation of voxel blocks. *Computer Graphics Forum*, 31(3):1315–1324, 2012.

[15] L. Ibarria, P. Lindstrom, J. Rossignac, and A. Szymczak. Out-of-core compression and decompression of large n-dimensional scalar fields. *Computer Graphics Forum*, 22(3):343–348, 2003.

[16] J. Iverson, C. Kamath, and G. Karypis. Fast and effective lossy compression algorithms for scientific datasets. In *Euro-Par Parallel Processing*, pages 843–856, 2012.

[17] J. Kunkel, A. Novikova, E. Betke, and A. Schaare. Toward decoupling the selection of compression algorithms from quality constraints. In *ISC High Performance International Workshops: DRBSD-I*, number 10524 in Lecture Notes in Computer Science, pages 1–12, 2017.

[18] S. Lakshminarasimhan. ISABELA version 0.2.1, June 2014. `http://freescience.org/cs/ISABELA/ISABELA.html`.

[19] S. Lakshminarasimhan, N. Shah, S. Ethier, S. Klasky, R. Latham, R. Ross, and N. F. Samatova. Compressing the incompressible with ISABELA: In-situ reduction of spatio-temporal data. In *Euro-Par Parallel Processing*, pages 366–379, 2011.

[20] D. Laney, S. Langer, C. Weber, P. Lindstrom, and A. Wegener. Assessing the effects of data compression in simulations using physically motivated metrics. In *ACM/IEEE International Conference on High Performance Computing, Networking, Storage and Analysis*, pages 76:1–12, 2013.

[21] P. Lindstrom. Fixed-rate compressed floating-point arrays. *IEEE Transactions on Visualization and Computer Graphics*, 20(12):2674–2683, 2014.

[22] P. Lindstrom. FPZIP version 1.1.0, June 2014. `https://computation.llnl.gov/casc/fpzip/`.

[23] P. Lindstrom. ZFP version 0.5.1, Mar. 2017. `https://computation.llnl.gov/casc/zfp/`.

[24] P. Lindstrom and M. Isenburg. Fast and efficient compression of floating-point data. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):1245–1250, 2006.

[25] W. A. Pearlman, A. Islam, N. Nagaraj, and A. Said. Efficient, low-complexity image coding with a set-partitioning embedded block coder. *IEEE Transactions on Circuits and Systems for Video Technology*, 14(11):1219–1235, 2004.

[26] A. Said and W. A. Pearlman. A new, fast, and efficient image codec based on set partitioning in hierarchical trees. *IEEE Transactions on Circuits and Systems for Video Technology*, 6(3):243–250, 1996.

[27] J. Schneider. HVQ version 1.31, Mar. 2010.

[28] J. Schneider and R. Westermann. Compression domain volume rendering. In *IEEE Visualization*, pages 293–300, 2003.

[29] S. K. Suter, J. A. I. Guitian, F. Marton, M. Agus, A. Elsener, C. P. E. Zollikofer, M. Gopi, E. Gobbetti, and R. Pajarola. Interactive multiscale tensor reconstruction for multiresolution volume visualization. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):2135–2143, 2011.

[30] D. Tao, S. Di, and F. Cappello. Significantly improving lossy compression for scientific data sets based on multidimensional prediction and error-controlled quantization. In *IEEE International Parallel and Distributed Processing Symposium*, pages 1129–1139, 2017.

[31] D. Tao, S. Di, H. Guo, Z. Chen, and F. Cappello. Z-checker: A framework for assessing lossy compression of scientific data. *International Journal of High Performance Computing Applications*, 2017. To appear. Available as arXiv:1707.09320.

[32] D. Taubman. High performance scalable image compression with EBCOT. *IEEE Transactions on Image Processing*, 9(7):1158–1170, 2000.

[33] D. Unat, T. Hromadka III, and S. B. Baden. An adaptive sub-sampling method for in-memory compression of scientific data. In *Data Compression Conference*, pages 262–271, 2009.

[34] A. Wegener. Universal numerical encoder and profiler reduces computing's memory wall with software, FPGA, and SoC implementations. In *IEEE Data Compression Conference*, page 528, 2013. Long version available as arXiv:1303.4994.