Reply to reviewers: "Finding the Goldilocks zone: Compression-error trade-off for large gridded datasets"

Jeremy Silver, Charlie Zender

October 28, 2016

We wish to thank the reviewers to taking the time to read the manuscript and provide feedback. We note that we have taken the challenge of major revision seriously and reworked the analysis to a much more fine-grained level, included a range of new and interesting results, remade all the figures, and restructured and rewritten much of the text. We believe that the reviewers' comments have helped to improve the manuscript and strengthen our findings.

Main changes

- Compression, errors and complexity are assessed at the variable-level, rather than the dataset-level (i.e. for a number of variables combined).
- We calculated a range of statistics on the individual variables, in order to improve our understanding of why certain variables compress well with one method or another.
- Some material was moved to a supplementary document.
- The introduction has been abbreviated as recommended.
- The Methods section was expanded to provide a clearer description of the layer-packing method.
- The discussion includes a brief review of related work.
- Additional description of the deflate and shuffle compression algorithms were added to the Methods section.
- All figures have been reworked.

Minor changes

- Variables are now chunked in a consistent manner for the different methods to improve comparability across compression methods.
- A minor error was found and corrected in the calculation of file sizes. The differences would have been very minor for the results in the original manuscript, since the file sizes were much larger than when doing the analysis on individual variables, but became apparent when working with the single-variable data files. The error was that the results were calculated based on "resident" rather than "actual" file size.

- Minor improvements were made to the layer-packing code, resulting in more stable treatment of non-finite values, avoiding rare cases of floating-point overflow, and more stable handling of dimensions.
- We ran the test suite on a variable of size 1.5 GB to examine the performance of the methods on larger datasets. This was included as an example referenced in the timing results, rather than adding it to the suite of variables presented in all results. This was mainly because, in the process of setting it up, the test suite was run many dozens of times; to accelerate the testing the variables considered were kept relatively small (the largest was about 65 MB).

1 Reviewer 1

1.1 General comments

1. *This paper addresses an important issue because data compression is very much needed to mitigate large data volumes in geophysical data. Treating the dimensions differently when applying lossy compression to gridded data makes a lot of sense.*

   We agree.

2. *Section 1 and 2 need some rearranging and improvement (more details are given below in "specific comments") in terms of introducing the ideas and terminology. It could be better to shorten the introduction and then really explain the methods well in section 2.*

   We have rearranged material in these sections given the feedback provided.

3. *The audience for this work may not be too familiar with compression techniques other than just using defaults in netCDF, so improving the explanations for the techniques would be helpful. (For example, defining a "deflate and shuffle" algorithm).*

   We have provided additional details as suggested.

4. *The paper's contribution should be clarified in the introduction (section 1). It is not clear to me whether "layer packing" is a new idea that is first presented here. (It is mentioned a bit more clearly in section 3).*

   Layer packing per say is not a new idea, and is the foundation for compression in the GRIB data format. However the idea of layer-packing is generalised here beyond two-dimensional slices. The work presented here is a test-of-concept for combining some of the better aspects of both GRIB and netCDF/HDF5 formats.
   The introduction and discussion reiterate these points.

5. *For this paper to really impact the broader geophysical data community, I feel that more details on the compression approaches need to be provided.*

   We have provided more details as recommended.

6. *More details on the datasets are needed to be able to understand why compression effects the each differently. Perhaps look at variables instead of multi-variable datasets?*

   This is an excellent suggestion and one that we have adopted. One of the main changes to the manuscript between the initial submission and this revision is that we examine compression in a variable-by-variable approach rather than as a whole-dataset approach.

This allows us to look at individual variables in terms of their compressibility, the "complexity" of the variable and error resulting from the lossy compression; this fine-grained approach allows for greater insight and a much larger sample size. As such the results section has been heavily revised.

1.2 Specific comments

1. *page 2, par. 1: For this audience, please give more explanation of the techniques. For example, please provide more explanation of how "deflate and shuffle" works (rather than just pointing to a reference).*

   We have introduced additional detail about these methods as recommended.

2. *page 2, line 22: "Linear packing with a single scale-offset parameter" – is discussed here but not well-defined. Note that "packing" is later defined in line 32. Then "scalar linear packing" on p.3. line 2. In general, the terminology used and defined in this paragraph is hard to follow in that it is sometimes defined after being used. (Also, is "linear packing with a single scale-offset parameter" the same as "scalar linear packing"?)*

   We have reviewed how the notation is introduced in order to improve readability.

3. *p.2, line 29: I'm not sure the audience will be familiar with "quantization" (like the audience for a CS publication would).*

   This has been clarified

4. *section 2.1.1 ("Layer packing") Here I would suggest providing more detail (maybe an example) – particularly if this approach is the main contribution of the paper. Rather than providing syntax details, consider defining/explaining the parameters (the reader may not be familiar with what these are) here.*

   In hindsight we agree that details about the algorithm itself are required, rather than syntax. We have moved the syntax to a supplementary section. The algorithm itself is outlined in the methods section.

5. *section 2.1.2, line 15: Explain what "level" means in the algorithm.*

   This has been explained.

6. *section 2.1.2, line 17: Explain a shuffle filter.*

   We have added additional details.

7. *section 2.3: Regarding the datasets listed, more information about the model source (other than acronym and reference) would be helpful - especially in interpreting the results. Without more details, I cannot really understand how the datasets differ and, therefore, why/how they would respond to compression differently. For example, the number of grid points are given - but does this number represent a domain on the entire globe for all datasets? The number of vertical levels is listed, but do all models simulate to the same height? What is the time dimension? Hourly? Monthly averages? Is the time dimension the same for each data set?*

The original description of these datasets was deliberately kept short, as this was not the main focus of the paper. We have compromised by abbreviating the description of the datasets to a table and moving the full descriptions of these datasets to the Supplementary Material section.
Regarding the question about why variables respond differently to compression, we believe that this has been solidly addressed in the analysis of the entropy of the data and exponent fields, which was made possibly by following the suggestion to shift the focus of the paper from compressing entire datasets to compressing individual variables.

8. *Fig 1: For compression results, I think it would be more intuitive/standard to compare to the uncompressed size (and have all ratios below 1.0). Also I don't understand the meaning of the comp./decomp. time in the left panel for uncompressed data.*

   The compression ratios are now defined in terms of the uncompressed size as suggested, and we have also moved to a more standard definition of the compression ratio (i.e. uncompressed size / compressed size, so that larger values represent greater compression). The compression times represent the time taken from the original data to the compressed file, whereas the decompression time is to unpack the layer-packed data. This has been clarified

9. *page 6, line 30: The paper could be much stronger with specific examples of individual variables and how affected by compression approach and choice of metric (e.g. by std. dev. or mean normalization). Since all results are averaged across datasets, this information is not available.*

   We agree and we have adopted the variable-level rather than dataset-level approach. We included examples of six variables (among a total of 255) in the Supplementary Material document as illustrations of the errors induced by the six lossy compression methods considered.

10. *Section 3: This section contains some useful information (and examples) about linear scaling and layer packing that would have been good to explain earlier in the paper when the concepts/algorithms are first introduced (and before the results are given).*

    We have given additional details about linear scaling and layer packing in the Methods section. Additional examples for illustrative variables appear in the Results section.

11. *More related lossy compression work on geophysical data should be mentioned for better context, for example: Hubbe, Wegener et al., ISC '13 (`http://link.springer.com/chapter/10.1007%2F978-3-642-38750-0_26`), Baker, et al., HPDC '14 (`http://dl.acm.org/citation.cfm?id=2600217`), Woodring et al., LDAV '11 (`http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6092314&tag=1`)*

    We have given more details about related lossy compression work in this field. We thank the reviewer for the suggested citations and have included some in the manuscript.

12. *Other competitive lossy compression algorithms for scientific data should probably be mentioned as many may be affected by differences in the variation across spatial dimensions for gridded data – this could be really interesting. Also many lossy compression methods for scientific data could eventually by incorporated into netCDF.*

    We have expanded the discussion to refer to other lossy compression algorithms for scientific data, formats beyond netCDF (e.g. based on image- and video-compression).

13. *Fig. 2: Because the differences between the datasets are not more thoroughly addressed, then it's unclear what conclusion to draw by comparing the SD and mean normalizations in Figure 2 (e.g., what is the takeaway point?). Basically, it seems that the two plots are quantitatively similar enough that both should be included only to illustrate a point, which I am not seeing. Can you clarify?*

    Both plots were included in order avoid the perception of a biased interpretation of the results. Normalization by the SD or the mean advantages one method or the other, however the conclusions are the same regardless of the normalization. We agree that including both plots does not add much value to the paper. We note that all the figures have been completely reworked.

14. *fig 3: Same comment as above, plus I am not sure what conclusion to draw given that some datasets compress better than others without a more clear understanding of dataset differences. I think looking at individual variables, rather than entire datasets would make it easier for the reader to understand the differences in the approaches.*

    As noted previously, we agree with the reviewer's comment and have redone the analysis to examine variables separately, rather than groups of variables clustered together as datasets.

1.3 Final thoughts

1. *I like the idea of treating spatial dimensions differently with lossy compression, and I think the authors could have really taken off with this concept and it explored it much more thoroughly. I question whether the contributions in this particular version are significant enough for a GMD paper.*

   The purpose of this study was to test the concept of layer-packing, in an attempt to combine some of the best aspects of the GRIB and netCDF/HDF5 data formats. We acknowledge that the results have not been conclusively in favour of the layer-packing with respect to bit-grooming, however we would argue that this is worth publishing all the same. This partly relates to the discussion of publishing "positive" versus "negative" results; if only "positive" findings are published, this will result in a great deal of time and effort being wasted within the scientific community in repeating superficially appealing experiments. As such, transferring this knowledge to the public domain has value. The geoscientific modelling and measurement community (e.g. the volume of data generated by satellite retrievals) relies heavily on these data formats, and it is important that their refinement is an ongoing process.
   Regardless of any ambiguity between the choice of bit-grooming or layer-packing, one clear result from this study is that simple linear packing typically results in *much* greater loss of precision than either of the two lossy methods discussed here. This is despite its widespread use.
   Other useful contributions include the focus on the error-compression trade-off, the finding that the normalized entropy of the exponent field can be used to help determine which compression method is most appropriate, and the idea (introduced in the discussion) that the changes in the normalized entropy of the data could be used to determine how many significant figures should be retained.

2 Reviewer 2

2.1 General comments

1. *This paper describes a variant of lossy encoding which leverages the multi-dimensional nature of many scientific datasets that have greater data variances along different axes. The axes with small variations in data values are labeled "thin dimensions" and the axes with large variations in values are labeled "thick dimensions". The datasets are then "layer packed" with a linear scaling algorithm in the thin dimensions, recording a scale & offset value for each coordinate in the thick dimension.*
   I think the insights into the "thick" and "thin" dimensions are the primary value of this paper, with the actual compression algorithm and results being less important, overall.

   Yes – one of the main things we are trying to do here is to assess whether treating different dimensions differently during gives much benefit over and above other methods that can be easily applied to such datasets. This is essentially trying to combine the best elements of GRIB and netCDF/HDF5.

2.2 Specific comments

1. *Applying the idea of thick & thin dimensions appropriately to other compression methods (such as the JPEG-2000 algorithm used in GRIB2) would be more valuable than just the idea of the simple scale & offset compression chosen.*

   We agree, and we spent a large amount of time trying to get this to work while preparing these revisions.
   In revising this work, we were able to run (after many technical hiccups) the same set of tests for GRIB/JPEG2000 compression as well (using 8, 12, 16 and 20 bits to represent the data). Our preliminary results showed that the JPEG2000 algorithm yields greater compression compared to the methods presented here for the same level of error; this echoes the findings of Caron (2014), which describe the efficient compression achieved. However like bit-grooming or layer-packing, JPEG2000 does not offer clear controls about the resultant errors and thus some experimentation (in setting the number of bits per value) is needed to avoid excessive loss of precision. We found that there was a large spread in the magnitude of the relative errors compared to the other methods considered. However the technical challenges required to convert a general netCDF field into GRIB format to be far in excess of what may be recommended to the average practitioner of geoscientific modelling. For this reason, and for the large spread in the compression and error result in the GRIB-compressed fields, and in order to keep the manuscript as focussed as possible, we chose not to include these results.

2. *Near the bottom of page 6, "for simplicity will have" should be corrected to "for simplicity we have".*

   Yes, well spotted. We have fixed this.

2.3 Reviewer 2's comments to Review 1

1. *Very nice review, much more detailed than mine. We seem to have homed in on the same insights: the differences in dimensions are the valuable part of the paper, and they aren't explored in enough detail to warrant a lot of enthusiasm in the current state of the*

paper. My current feeling is a very "weak" accept, and I would prefer to ask for further exploration of the dimension ideas.

Following the suggestions from Reviewer 1, the analysis and results have been considerably expanded and the revised manuscript offers further perspectives into the relationship between lossy compression, the resulting error and underlying complexity of the data.

3 Reviewer 3

3.1 Summary

The paper introduces a "layer packing" lossy compression technique that takes advantage of the minimal horizontal variations in geoscience data relative to the larger variations across vertical dimensions. The layer packing technique is compared against many widely used lossless and lossy compression techniques and evaluated based on accuracy and time to solution. Layer packing is found to be beneficial in some cases while not in others, leading to the conclusion that care must be taken to evaluate whether lossy compression is worth the risk.

3.2 General Comments

1. *The paper makes a good first attempt to evaluate the layer packing technique, but the paper would benefit from an additional revision. First, it's not clear what the paper is contributing. The authors state that the technique is used in GRIB (page 7, section 3) but that the evaluation was not possible due to relative error not being reported. Since the technique is not new, then the only contributions of the paper are the announcement of the general availability of the new non-GRIB tools, as well as the modestly detailed evaluation of the many compression techniques.*

   The geoscientific modelling and remote-sensing community has to deal with the ever-growing volume of data generated. As such, it is important that the storage methods are reviewed in terms of the trade-off between compression, error and read/write times.
   We have tried to avoid debate about data formats. Both have an important roles; the geoscientific community relies heavily on netCDF/HDF5, and GRIB remains the format of choice in many operational meteorological centres. Despite its excellent compression performance, GRIB can be regarded as less user-friendly.
   The GRIB layer-packing is restricted to two-dimensional slices, whereas the layer-packing described here can operate on arbitrary hyperslices. The work presented in this manuscript aims to generate discussion about ways of incorporating the best of both methods.
   With reference to the comment from Page 7, Section 3: "Caron (2014) estimated that GRIB2 files are on average 44% of the size of the equivalent deflate-compressed netCDF-4 files (n.b. relative errors were not reported, which limits the comparison)". The intended meaning was that the study of Caron (2014) reported the compression ratio, but not the relative errors, which makes it difficult to place the Caron (2014) results with those of this study.
   The revisions to the original manuscript, focusing the analysis on the compressibility, errors and complexity of individual variables offers additional insights into these relationship and we believe adds substantially to the value of the paper.

3.3 Specific Comments and Technical Corrections

1. *The title, though catchy, is overloading the term "Goldilocks Zone" – the region around a star where perhaps liquid water might be found on a planet's surface. The title after the colon is clear on its own.*

   We have abbreviated the title, which as already been through several iterations.

2. *Page 2, line 3: "NetCDF" starts the last sentence on the line, though it should be "netCDF" for consistency.*

   We have revised for consistency of this term.

3. *Page 2, line 5: Why are three references necessary to describe the "deflate" compression method? Throughout the paper, be consistent with terms. scale-offset vs scale and offset. linear-packing vs linear packing.*

   Additional description of the deflate and shuffle algorithms has been added as suggested by Reviewer 1. We have reconsidered the references in this section. We have also reviewed the usage of the terms mentioned to improve the consistency of the manuscript.

4. *Page 3, line 30: I would suggest adding that ncdump is a command-line utility from the netCDF package because it might not be common knowledge. The paper introduces the "ncpacklayer" program and also uses other "nc"-prefixed tools from the NCO suite. For example, perhaps the following: "...(following the output format for the netCDF command-line utility ncdump)..."*

   Yes, this is correct, thanks for pointing this out. We have clarified this point.

5. *Page 3 (section 2 in general): More detail could be spent on the layer packing technique itself; the many monospaced examples of section 2 don't substantially add to the narrative and instead come across like a tutorial or README.*

   We have expanded the description of the algorithm itself. To keep the article short and concise, we have moved these details to an appendix.

6. *Page 4, line 11: run-on sentence*

   Thanks for pointing this out. This has been corrected.

7. *Page 4: The dollar symbol "$" is not explained, though I think you meant for it to refer to a shell variable syntax.*

   Yes, this is correct. This has been clarified.

8. *Page 5, Section 2.3: If I do the math correctly, the size of the datasets are (1) 962MB, (2) 267MB, (3) 68MB, (4) 613MB, (5) 30MB, and (6) 717MB. The rationale for the proposed compression is the growing volume of data in the geosciences, though none of these datasets are over a gigabyte in size. Compression of a multi-gigabyte dataset would make the argument more compelling, because datasets of such size will become more commonplace. Writing large datasets to disk as they are computed is a challenging problem and it would be nice to evaluate whether compressing large datasets is a viable option as they are generated. General comment about all Figures: Consider labeling the left and right panes of each figure as (a) and (b). For example, page 6, paragraphs starting on lines 9 and 17 sound too similar since Figure 1 is showing different things but is referred*

to in the text in the same way. It would be more clear to say something like "Figure 1A shows..." and "Figure 1B presents..."

The point about the magnitude of the file size is quite reasonable. We ran the test suite on variable of size 1.5 GB to examine the performance of the methods on larger datasets. This was included as an example referenced in the timing results, rather than adding it to the suite of variables presented in all results. This was mainly because, in the process of setting it up, the test suite was run many dozens of times; to accelerate the testing the variables considered were kept relatively small (the largest was about 65 MB).
However by the same token, the analysis for the revised manuscript has been done on individual variables alone, so the basic unit of study has become much smaller. While this might not impress those working with terabyte-scale data, it allows for greater insight into the methodology itself.
Regarding the figures, some of these have been moved to a supplementary material section. All panel plots now have labels (a), (b), etc., as suggested.

9. *Page 7: Starting on this page, for some reason all references to "figure 3" are lower case.*

   Thanks for pointing this out – it has been fixed.

10. *Page 8: Figure 1: The red and orange colors are too similar, though their position is clear from the legend.*

    All the figures have been thoroughly reworked. The color scheme in question no longer appears.

11. *Page 8, Figure 1, right panel: What does it mean to have the first column as "uncompressed" time since everything is normalized to DEFLATE? Was it the time to generate the data? Was it the time to copy the file?*

    Yes, in hindsight this wasn't very clear. It was effectively the time to copy the data. This bar is not included it in the revised manuscript. Thanks for drawing attention to it.

12. *Page 8, line 4: The reference to the HDF Group is used as an in-text citation as "(Group, 2016)". It would be best to fix your citation to not use HDF Group as a first/last name pair. See also your references on page 13, line 17.*

    Thanks for pointing this out. The default behaviour of the reference manager should have been over-ruled. This has been corrected.

13. *Page 9, line 1: run-on sentence*

    Thanks for pointing this out. It has been corrected.

14. *Page 10, Figure 3 caption: capitalize the Figure 1 and Figure 2 references.*

    This has been made more consistent.

15. *Page 11, line 6: misspelled "considered" – please consider a full spell check.*

    This has been fixed and we will ensure to run the spell checker again before resubmitting.

References

Caron, J. (2014). Converting GRIB to netCDF-4: Compression studies. `www.ecmwf.int/sites/default/files/elibrary/2014/13711-converting-grib-netcdf-4.pdf`, Last accessed: 2016-06-17. Presentation to the workshop "Closing the GRIB/netCDF gap", held at European Centre for Medium Range Weather Forecasts (ECMWF) at Reading, UK, 24-25 September 2014.

~~Finding the Goldilocks zone:~~ Compression-error trade-off for large gridded datasets

Jeremy D. Silver[1] and Charles S. Zender[2]

[1]School of Earth Sciences, University of Melbourne, Australia

[2]Departments of Earth System Science and of Computer Science, University of California, Irvine, USA

Correspondence to: J. D. Silver (jeremy.silver@unimelb.edu.au)

Abstract. The netCDF-4 format is widely used for large gridded scientific datasets, and includes several compression methods: lossy linear scaling and the non-lossy deflate and shuffle algorithms. Many multidimensional geoscientific datasets exhibit considerable variation over one or several spatial dimensions (e.g. vertically) with less variation in the remaining dimensions (e.g. horizontally). On such datasets, linear scaling with a single pair of scale and offset parameters often entails considerable loss of precision. We ~~propose a method (termed "layer packing")~~ introduce an alternative compression method called "layer-packing" that simultaneously exploits lossy linear scaling and lossless compression. ~~Layer packing~~ layer-packing stores arrays (instead of a scalar pair) of scale and offset parameters.

An implementation of this method is compared with ~~existing compressiontechniques~~ lossless compression, storing data at fixed relative precision (bit-grooming) and scalar linear packing in terms of compression ratio, accuracy ~~,~~ and speed. ~~Layer packing produces typical errors of 0.01-0.02of the standard deviation within the packed layer, and yields files roughly 33smaller than the lossless deflate algorithm. This was similar to~~

When viewed as a trade-off between compression and error, layer packing yields similar results to bit-grooming (storing between 3 and 4 significant figures~~per datum. In the six test datasets considered, layer packing demonstrated a better compression/error trade-off than storing 3-4 significant digits in half of cases and worse in the remaining cases, highlighting the need to compare lossy compression methods in individual applications. Layer packing preserves substantially more~~). Bit grooming and layer-packing offer significantly better control of precision than scalar linear packing~~, whereas scalar linear packing achieves greater compression ratios~~. Relative performance, in terms of compression and errors, of bit-groomed and layer-packed data were most strongly predicted by the entropy of the exponent array, and lossless compression was well predicted by entropy of the original data array. Layer-packed data files must be "unpacked" to be readily usable. ~~These~~ The compression and precision characteristics make layer-packing a competitive archive format for many ~~geophysical~~ scientific datasets.

Keywords. netCDF-4; HDF5; Lossy compression; Data storage format

1 Introduction

The volume of both computational and observational geophysical data has grown dramatically in recent decades, and this trend is likely to continue. While disk storage costs have fallen, ~~strong~~ constraints remain on data storage. Hence, in practice, compression of large datasets continues to be important for efficient use of storage resources.

Two important sources of large volumes of data are computational modelling and remote-sensing (principally from satellites). These data often have a "hypercube" structure, and ~~storage~~ are stored in formats such as HDF5 (http://www.hdfgroup.org), netCDF-4 (http://www.unidata.ucar.edu/netcdf) and ~~GRIB~~ GRIB2 (http://rda.ucar.edu/docs/formats/grib2/grib2doc/). Each of these has their own built-in compression techniques, allowing data to be stored in a compressed format, while simultaneously allowing access to the data (i.e. incorporating the compression/decompression algorithms into the format's API). These compression methods are either "lossless" (i.e. no precision is lost) or "lossy" (i.e. some accuracy is lost).

~~The~~ In this study, we examine the advantages and trade-offs of in allowing for different treatment of dimensions in the compression process. One motivation for this work is improving the lossy compression ratios typically achieved with HDF5~~and netCDF-4 formats~~ /netCDF4, so they are more comparable with impressive compression achieved by GRIB2. Records in GRIB2 are strictly two-dimensional and this format allows for only a limited set of predefined metadata. However GRIB2 offers excellent compression efficiency (Caron, 2014) , built upon the JPEG image compression methods; it is therefore well-suited for operational environments. By contrast, HDF5/netCDF4 provide a highly flexible framework, allowing for attributes, groups, and user-defined data types. ~~Variables can be defined with as many as 32 or 1024 dimensions (respectively for the two formats); the relatively free framework to define meta-data allows such formats to be self-documenting and applicable for many contexts. NetCDF-4 is built upon HDF5, and inherits most of its capabilities. Both~~ This format is well suited for more experimental environments and allows introduction of user-defined parameters to guide and improve compression. We use this flexibility to develop a lossy compression algorithm, termed "layer-packing", which combines desirable features from both GRIB2 and HDF5/netCDF4.

Layer-packing is a variant of compression via linear scaling that exploits the clustering of data values along dimensional axes. We contrast this with other compression methods (rounding to fixed precision, and simple linear scaling) that are readily available within the netCDF4/HDF5 framework. The performance is quantified in terms of the resultant loss of precision and compression ratio. We also examine various statistical properties of datasets that are predictive for their overall compressibility and the relative performance of layer-packing or rounding to fixed precision.

2 Methods

This section outlines the implementation of the layer-packing and ~~netCDF-4 include the~~ -unpacking methods, the storage format (in the implementation described), the test data sets, evaluation metrics and the performance of the compression methods on the test cases considered.

2.1 Compression algorithms

2.1.1 Deflate and shuffle

The "deflate" ~~compression algorithm (Deutsch, 2008; Ziv and Lempel, 1977, 1978) and a shuffle filter~~ algorithm (Deutsch, 2008) is a widely-used, lossless compression method, available within most modern installations of the netCDF4 or HDF5 API. The deflate algorithm breaks the data into blocks, which are compressed via the LZ77 algorithm (Ziv and Lempel, 1977, 1978) combined with Huffman encoding (Huffman, 1952) . The LZ77 algorithm searches for duplicated patterns of bytes within the block. The Huffman coding step involves substituting frequently used "symbols" with shorter bit-sequence and rarer symbols with longer bit-sequence. ~~These are "lossless" compression techniques, in that no precision is lost. By contrast, "linear scaling" is a "lossy" compression method (implying a loss of accuracy), which rounds the~~

When used in the netCDF4/HDF5 framework, the deflate algorithm is often applied together with the shuffle filter (HDF5 Group, 2002) . The shuffle filter does not compress data as such, but instead changes the byte ordering of the data; the first byte of each value is stored in the first chunk of the data stream, the second byte in the second chunk, and so on. This tends to improve the compressibility of the data, particularly if there is autocorrelation in the data stream (i.e. data that are close in value tend to appear close together).

In all of the following work, we have used the deflate algorithm and the shuffle filter together, and are henceforth referred to collectively as DEFLATE for brevity. In the results presented below, compression via DEFLATE was performed with the `ncks` tool of the NCO bundle (Zender, 2008) . The compression level (set taking values between 1 & 9, where 1 means fastest and 9 means greatest compression) dictates the amount of searching for duplicated patterns that is performed within the LZ77 step. For all applications (either as an independent method or in combination with the methods outlined below), the same compression level was used within DEFLATE (namely, level 4).

When using the netCDF4 framework to compress a variable, the user must define the size of the "chunk" within which compression occurs. In all the results presented here (both for DEFLATE and for the other compression methods), the chunk size was equal to the layers packed using layer-packing (see Section 2.1.3). The same analyses were performed setting the chunk sizes to encompass the whole variable, and the conclusions reached in this article were essentially the same.

2.1.2 Scalar linear packing

One can compress a field using a lower-precision value (e.g. as two-byte integers rather than four-byte floating point numbers) and a linear transformation between the original and compressed representations; this process is termed "packing". In the common case of representing four-byte floats as two-byte integers, there is already a saving of 50% relative to its original representation (ignoring the small overhead due to storing parameters involved in the transformation). In this case, a range of transformations are readily available. We call this "scalar linear packing" (or LIN, for short) and it is the standard method of packing in the geoscience community. Its attributes are defined in the netCDF User Guide (Unidata, 2016) and it has a long and wide tradition of support, including automatic interpretation in a range of netCDF readers.

Scalar linear packing to convert floating-point data to ~~discrete values within a defined range.A common usage of linear scaling is to represent variables as (signed or unsigned)~~an unsigned two-byte ~~integers, which allows for $2^{16} = 65536$ different values. If applied to a four-byte floating point (i.e. single precision)array, this use of linear scaling cuts data usage by~~ integer representation is outlined below:

$maxPackedValue = 2^{16} - 1 = 65535$

$vMin$ = minimum value of $data$

$vMax$ = maximum value of $data$

$add\_offset = vMin$

$scale\_factor = (vMax - vMin)/maxPackedValue$

$packed = \mathrm{uint16}((data - add\_offset)/scale\_factor)$

where $vMin$, $vMax$, add_offset and $scale\_factor$ are scalar floating point values, $data$ is a floating point array and $packed$ is an unsigned two-byte integer array of the same dimension as $data$, and the function $\mathrm{uint16}(\cdot)$ converts from floating point to two-byte integer. Care must be taken in handling infinite, not-a-number or undefined values (such details are omitted here). The value of $maxPackedValue$ listed uses all values in the two-byte integer range to represent floats; one can choose a different value of of $maxPackedValue$, leaving some two-byte values for the special floating-point values. The values of add_offset and $scale\_factor$ must be stored along with $packed$ to enable the reverse transformation:

$unpacked = \mathrm{float}(packed) \times scale\_factor + add\_offset$

Data packed with this method often can be compressed substantially more than the 50% noted above. This is done by applying DEFLATE to the packed data; this was done for all datasets compressed with LIN here. In this study, compression via DEFLATE was performed with the `ncpdq` tool of the NCO bundle (Zender, 2008) . ~~When the deflate and shuffle filters are also applied, the savings are typically much greater.~~

2.1.3 Layer-packing

In many applications in the geophysical sciences, a multidimensional gridded variable ~~may range~~ varies dramatically across one dimension while exhibiting a limited range of within slices of this variable. Examples include:

- variation in atmospheric density ~~,~~ and water vapour mixing ratio with respect to height
- variation in ocean temperature ~~,~~ and current velocity with respect to depth
- variation in atmospheric concentrations of nitrogen dioxide with respect to height and time

We will use the term “thick dimensions” to denote those dimensions that account for the majority of the variation in such variables, “thin dimensions” to denote the remaining dimensions; in the case of the first example above (assuming a global grid and a geographic coordinate system), the vertical dimension (pressure or height) is thick, and the horizontal dimensions (latitude, longitude) and time are thin. We will use the term “thin slice” to describe a slice through the hypercube for fixed values of the thick dimensions. Note that there are cases with multiple thick dimensions, such as the third example noted above.

~~Linear-packing with a single scale-offset pair~~ Scalar linear packing applied to such gridded variables ~~results~~ will result in a considerable loss of accuracy~~across thin slices of such variables~~. This is because in order to cover the scale of variation spanned by the thick dimensions, few discrete values will be spanned by the individual thin slices. ~~At least two distinct methods can achieve a better compromise between data compression and loss of accuracy. First, one~~ To reduce loss of precision within the thin slice, one can store for each variable *arrays* of scale-offset pairs, with size corresponding to the thick dimensions only. This is the key innovation of the layer-packing technique (or LAY for brevity). Layer-packing, as discussed here, was implemented via the following algorithm (described here in pseudocode):

```
for $var$ in $vars$ do
  if ($var$ in $splitVars$) and intersection(dimensions[$var$],$splitDims$) is not None then
    $theseSplitDims$ = intersection(dimensions[$var$],$splitDims$)
    $theseDimLens$ = lengths of $theseSplitDims$
    $iSplitDim$ = indices of $theseSplitDims$ in dimensions[$var$]
    $splitDimIdxs$ = all possible index combinations for $theseSplitDims$
    $nSplits$ = number of combinations given in $splitDimIdxs$
    $nDim$ = length of dimensions[$var$]
    $indices$ = list of length $nDim$, with each element equal to Ellipsis
    $data$ = the data array
    $packed$ = array of zeros, type is uint16, with the same shape as $data$
    $add\_offset$ = array of zeros, type is float, with dimensions given by $theseDimLens$
    $scale\_factor$ = same as $add\_offset$
    for $iSplit$ in range(0,$nSplits$) do
      $indices[iSplitDim] = splitDimIdxs[iSplit]$
      $thinSlice = Data[indices]$
      $vMin$ = minimum value of $thinSlice$
      $vMax$ = maximum value of $thinSlice$
      $add\_offset[splitDimIdxs[iSplit]] = vMin$
      $scale\_factor[splitDimIdxs[iSplit]] = (vMax - vMin)/maxPackedValue$
      $packed[indices]$ = uint16($(thinSlice - add\_offset)/scale\_factor$)
    end for
    write $packed$, $add\_offset$, $scale\_factor$ to file
  else
    write the $data$ array to file
  end if
```

end for

The two-byte representation halves the storage cost of the array itself, however arrays of scale factors and linear offsets must also be stored and this adds to the total space required. Compression is generally significantly improved by applying DEFLATE and this was done for all datasets presented here.

Further details about the implementation are given in the Supplementary Material document. The code to perform layer-packing described in this article was written as stand-alone command-line tools in Python (v. 2.7.6). These are freely available on https://github.com/JeremySilver/layerpack. Beyond the standard Python installation, it requires the `numpy` and `netCDF-4` modules be installed.

2.1.4 Bit grooming

One can store the data at a fixed precision (i.e. a chosen number of significant digits, or NSD). This method is known as "bit-grooming" and is detailed by Zender (2016) and implemented in the NCO package (Zender, 2008). If bit-groomed data remain uncompressed in floating-point format this coarsening will not affect the file size, however the application of the deflate/shuffle algorithms will in general improve the compressibility of the coarsened data, ~~due to quantization to~~ as they will be represented by a smaller number of ~~values.~~ discrete values. For further explanation, we must briefly summarize how floating-point numbers are represented by computers.

~~Second, one can represent the field using a lower-precision value (e. g. as two-byte integers rather than four-byte floating point numbers) and an agreed transformation between the original and compressed representations; this process is termed "packing". In the common case of representing four-byte floats as two-byte integers, there is already a saving of 50relative to its original representation (ignoring any overhead due to storing parameters involved in the transformation) . In this case, a range of transformations are readily available. The simplest is to use linear transformation with a single scale-offset pair, thereby avoiding the thick/thin dimension distinction. We call this "scalar linear packing" and it is the standard method of packing in the geoscience community. Its attributes are defined in the netCDF User Guide (Unidata, 2016) and it has a long and wide tradition of support, including automatic interpretation in a range of netCDF readers. Another option is to store for each variable *arrays* of scale-offset pairs, with size corresponding to the thick dimensions only. We will refer to this technique as "layer packing". In this article, we compare the merits of these lossy compression methods alongside established compression techniques. We do not consider packing procedures using application-specific transformations; although these may be efficient for particular applications, they are of less general interest and if they involve storing normalization factors (i.e. parameters for use in the transformation) then their performance is likely to perform similarly tothe layer packing described above.~~ Single-precision floating-point numbers occupy 32 bits within memory. Floating-point numbers are represented as the product of a sign, significand and an exponential term:

$$\text{datum} = \text{sign} \cdot \text{significand} \cdot \text{base}^{\text{exponent}}$$

The IEEE standard specifies that the sign accounts for 1 bit, the significand (also known as the mantissa) 23 bits and the exponent 8 bits. The base is 2 by convention. The sign is an integer from the set $\{-1,1\}$, the significand is a real number in the range $[0.0,1.0)$ and the exponent is an integer between -128 and +127; see, for example, (Goldberg, 1991) for further details.

~~In Section 2, we compare the performance of bit grooming, scalar linear packing, layer packing, and deflate methods. These are assessed in terms of the compression savings and loss of precision. The choice of the compression methodology depends on other factors such as read/write speeds, portability and flexibility. These considerations are discussed in Section 3~~Bit grooming quantizes[1] data to a fixed number of significant digits (NSD) using bitmasks, not floating point math. The NSD bitmasks alter the IEEE floating point mantissa by changing to 1 (bit setting) or 0 (bit shaving) the least significant bits that are superfluous to the specified precision. Sequential values of the data are alternately shaved and set, which nearly eliminates any mean bias due to quantization (Zender, 2016) . To guarantee preserving 1–6 digits of precision, bit grooming retains $5, 8, 11, 15, 18$ and 21 explicit mantissa bits, respectively, and retains all exponent bits.

3 ~~Methods~~

~~This section outlines the implementation of the layer-packing and unpacking methods, the storage format (in the implementation described), the test data sets, evaluation metrics and the performance of the compression methods on the test cases considered~~In the following we compared storing 2, 3, 4 and 5 significant digits; these are denoted NSD2, NSD3, NSD4 and NSD5, respectively. Similar to LIN and LAY, DEFLATE was applied together with rounding. In this study, compression via bit-grooming was performed using the `ncks` tool within the NCO package (Zender, 2008) .

2.1 ~~Software implementation and storage format~~Datasets

2.1.1 ~~Layer packing~~

~~The code to perform layer-packing described in this article was written as stand-alone command-line tools in Python (v.2.7.6). These are freely available on . Beyond the standard Python installation, it requires the `numpy` and `netCDF-4` modules be installed. The compression is performed as follows: Other optional flags allow for increased verbosity (`-V`), over-writing existing output files (`-O`) and defining the DEFLATE compression level (`-L`).~~

~~The mandatory `-d` flag is followed by a comma-separated list of the thick dimensions. The optional `-v` flag is followed by a comma-separated list of variables to pack. The default is to pack all variables defined along any of the thick dimensions listed.~~ In the ~~output file (in this example `packed.nc`) each variable that is packed (e.g.`var1`) is replaced by a trio of variables containing the arrays of packed values, scale factors and offsets. In this example, these are termed `var1SUBSCRIPTNBSUBSCRIPTNBsh`~~ ~~`var1SUBSCRIPTNBSUBSCRIPTNBscale`~~ following tests, we compared a total of 255 variables from six datasets. Each variable was extracted individually to file as uncompressed netCDF, and ~~`var1SUBSCRIPTNBSUBSCRIPTNBoffset`, with data type unsigned short~~ the file was then compressed using the methods described, allowing for computation of compression

[1]The process of quantization means mapping, in this case via a process similar to rounding, from a large set of possible inputs (in this case the full set of real numbers representable as floating point values) to a smaller set (in this case those floating points defined to reduced precision desired by the user).

~~significant figures (respectively). The following yields three significant digits (NSD3). `ncks -4 -L4 -ppc vars=3 in.nc out.nc` LSSCA~~

Table 1. Summary of the datasets used in this study. Abbreviations: ID = index, NWP = numerical weather prediction, CTM = chemistry-transport model, Dims = Dimensions of the variable, TS Dims = Dimensions of the thin slice, # vars = number of variables per dataset. The dimension sizes are indicated as: n_x = length of the east-west dimension, n_y = length of the north-south dimension, n_z = length of the vertical dimension, n_t = length of the time dimension and $n_{x'}$ = length of the generalized horizontal coordinate dimension (used for the unstructured grid in last dataset only).

and error metrics described below. The datasets are summarised in Table 1. Further details are provided in the online Supplementary Material section.

The variables chosen from these datasets were those with the largest number of data-points overall. For example in datasets 2-5, variables without a vertical coordinate were not considered in the analysis, since these account for only a small fraction of the total data. A small number of the variables that would otherwise be included (based on the dimensions alone) were excluded due to the occurrence in seemingly random data (i.e. ~~two-byte) integer, floating-point and floating-point, respectively. Suppose the original definition of `var1` is (following output format for the command line utility `ncdump`) : then the corresponding trio will have dimensions as follows: In other words, the scale and offset arrays have one element per thin slice. Data remain in netCDF format in this packed format and retain all their attributes. Data can be unpacked as follows: The `-d` and `-v` flags are not used, since this information is contained in the trios of packed arrays.~~

2.1.1 ~~Methods compared~~

of all magnitudes) in regions of the array where values were not defined (in the sense of sea-surface temperatures over land points), which we believe should have been masked with a fill-value; the rationale for excluding these variables is first, that these regions did not appear to contain meaningful data and that the extreme range of the seemingly-random data led to gross outliers in the distribution of error statistics for LIN and LAY in particular.

~~We will compare layer packing with a number of other compression methods, which in each case use command-line tools from the NCO bundle (Zender, 2008) . The formats compared, the commands used to achieve these and the capitalized abbreviations, are listed below. UNCOMPRESSED: Uncompressed netCDF `ncks -3 in.nc out.nc` DEFLATE: Deflate compression (level 4) with shuffle filter `ncks -4 -L4 in.nc out.nc` NSD2, NSD3, NSD4, NSD5: Deflate compression (level 4) with shuffle filter, and bit grooming storing~~

2.2 ~~Tests~~Error and compression metrics

The methods are compared with two metrics. The first relates to the compression efficiency~~, relative to DEFLATE above (deflate compression with shuffle filter)~~. Compression ratios are defined ~~relative to the file sizegenerated by DEFLATE (i.e.smaller is better)~~as (uncompressed size)/(compressed size), and as such larger values indicate greater compression. The second metric relates to the accuracy (or, seen another way, the error) of the compressed data relative to the original data. The error of UNCOMPRESSED is zero (as it is the reference data), as is DEFLATE since it ensures lossless compression. The remaining methods cause some loss of precision.

~~These errors can be summarised in different ways. In~~ Error was quantified by the root mean-square difference between the original and the compressed variables. However in order to compare results across variables with different scales and units, the errors must be normalized somehow. ~~A single normalization factor per variable can be used, however this has the disadvantage that the results will be heavily influenced by variation across the thick dimension; for example, if a field ranges across several orders of magnitude~~ We considered four different normalization methods, which emphasized different aspects of the error profile. The errors were normalized either by the standard deviation or the mean of the original data, and these were either calculated separately per thin slice or across the entire variable – the rationale is as follows.

When normalizing by the mean (of the entire variable, or of the thin-slice), variables with a low mean-to-standard-deviation ratio (e.g. potential vorticity) will show larger errors using the layer-packing compression. Whereas when normalizing by the standard deviation, variables with a high mean-to-standard-deviation ratio will show larger errors using the bit-grooming compression (e.g. ~~values of $O(10^0)$ at one end of the thick dimension and~~ atmospheric temperatures stored with units of K, concentrations of well-mixed atmospheric trace gases such as CO_2 or CH_4).

If we calculate the ratio of the RMSE to normalization factor (i.e. the mean or standard deviation) per thin slice, and then average across the normalized errors (n.b. there is thus one per thin slice) the resulting metric will be more sensitive to large relative errors within subsections of the data array. The alternative is to calculate the normalization factor across the whole variable, and the resulting metric will be more reflective of relative errors across the entire data array. This may be understood in the context of a hypothetical thee-dimensional array, with values ranging from $O(10^4)$ ~~at the other) , then errors will be highly non-uniform across the thick dimension.As such, we choose to normalize errors within each thin slice of a given variableby the standard deviation or mean of~~ to $O(10^0)$ across the ~~original values across that thin slice; the rationale for~~ vertical dimension and a mean value of $O(10^3)$, and errors roughly uniform of $O(10^{-1})$; if relative errors (normalizing by the ~~standard deviation or the mean is discussed in section 2.4 below. For~~ mean) are calculated for the whole array then they will be $O(10^{-4})$, whereas if calculated across each thin slice separately they will range from $O(10^{-1})$ to $O(10^{-5})$, and may have a mean of $O(10^{-2})$. The case of uniform errors is most likely to arise for LIN, whereas bit-grooming guarantees precision for each individual datum and layer-packing for each thin slice~~of a given variable, the standardized error is defined as the ratio of the root mean-squared error to the standard deviation (or the mean); for a given variable , the standardized error is defined as the mean standardized error averaged across all thin slices. The error for LSARRAY is computed based on the unpacked contents of the file~~.

2.3 ~~Datasets~~

2.3 Complexity statistics

In order to make sense of which variables compress well or poorly with different methods, a range of statistics were calculated for each variable. Most of these statistics were calculated over two-dimensional hyperslices of the original data arrays and then the value for an individual variable was taken as the average over these hyperslices. A full list of the statistics calculated is given in the Supplementary Material document.

The ~~tests described above were applied to the following datasets: `eiSUBSCRIPTNBmnthSUBSCRIPTNBanSUBSCRIPTNBplSUBS` ERA-Interim reanalysis data (Dee et al., 2011) . Dimensions: 121×240 grid-points in the horizontal, 37 vertical layers, 16 time-points.14 variables with these four dimensions.Thick dimensions chosen to be the time and vertical level. `mozart4geos5SUBSCRI` A limited area subset from global MOZART model output (Brasseur et al., 1998) . Dimensions: 9×10 grid-points in the horizontal, 56 vertical levels, 172 time-points. 77 variables with these four dimensions. Thick dimension chosen to be the vertical level. `wrfoutSUBSCRIPTNBd03SUBSCRIPTNB2013-01-24SUBSCRIPTNB07:00:00` WRF model output (Skamarock et al., 2005) . Dimensions: 165×140 grid-points in the horizontal, 32 vertical levels, 1 time-point. 23 variables with these four dimensions, which constitute 85of the data stored in the file. Thick dimension chosen to be the vertical level. `MERRA300.prod.assim.inst3SUBSCRIPTNB3dSUBSCRIPTNBasmSUBSCRIPTNBCp.20130601.nc` MERRA reanalysis product (Rienecker et al., 2011) . Dimensions: 144×288 grid-points in the horizontal, 42 vertical levels~~two most informative statistics that arose from this analysis were based on the entropy of either the original data field or the corresponding exponent field (i.e. the decomposing the data array into significand and exponent, and then calculating the entropy of the exponent array). The entropy is a measure of statistical dispersion, based on the frequency with which each value appeared in each dataset. Let us denote as $\mathrm{P}(x_i)$ the proportion of the array occupied by each unique value x_i. For an array containing discrete values $X = \{x_i\}_{i=1}^{k}$, the entropy was defined as

$$H(X) = \mathbb{E}[-\log_2(\mathrm{P}(X))] = -\sum_{i=1}^{k} \mathrm{P}(x_i)\log_2(\mathrm{P}(x_i)) \tag{1}$$

For an array of k, the entropy has a maximum value of $\log_2(k)$, which will arise if all values are unique. For single-precision arrays of size $2^{32} \approx 4.29 \times 10^9$ or larger, the maximum entropy is equal to 32.

In order to normalize for these limitations to the entropy of a finite dataset, for each case the entropy will be normalized by the maximum theoretical value attainable for that dataset, which was taken to be $\log_2(K)$, where K is the number of elements in the thin slice (in each case $K << 2^{32}$). In the case of the entropy of the exponent array, the normalization was based on the $\min(\log_2(K), 8)$, since the maximum entropy of an 8-bit field (recall, 8 ~~time-points. 11 variableswith these four dimensions. Thick dimension~~ bits are used for the exponent of a floating point) is 8.

It was found that some of the datasets compressed significantly using DEFLATE only. This was often due to a high proportion of zero or "missing" values. Variables were classified as either "sparse" (highly compressible or otherwise relatively simple) or "dense" (all other variables). Sparse variables were chosen to be ~~the vertical level. `dstmch90SUBSCRIPTNBclm.nc` Output~~

~~of the mineral Dust Entrainment And Deposition (DEAD) model (Zender et al., 2003) . Dimensions: 94 × 192 grid-points in the horizontal, 28 vertical levels, 1 time-point. 15 variables with these four dimensions, together accounting for 87of~~ those satisfying any one of the following conditions: the ~~data stored in the file. Thick dimension chosen to be the vertical level.famipc5SUBSCRIPTNBne30SUBSCRIPTNBv0.3SUBSCRIPTNB00003.cam.h0.1979-01-L5.nc CAM-SE model output (Dennis et al., 2012) .Dimensions: 48602 grid-points in the horizontal, 30 vertical levels, 1 time-point. 123 variables with these three dimensions, together accounting for 87of the data stored in the file. Thick dimension chosen to be the vertical level.~~ compression ratio is greater than 5.0 using DEFLATE, the fraction of values equal to the most common value in the entire variable is greater than 0.2, and the fraction of hyperslices where all values were identical is great than 0.2. These definitions were somewhat arbitrary and other classifications may be optimal, but it is seen (e.g. in Figures 1C, 3A and 3B) that sparse variables do not always follow the same pattern as dense variables. Of the 255 variables in total, 181 were classified as dense. The breakdown among the different categories is given in Table 1 in the Supplementary Material document.

2.4 ~~Results~~Compression and error results

Figure 1~~shows the compression ratios for the six test data sets. As noted above, compression ratios are defined by normalizing by the file sizes from DEFLATE (deflate and shuffle). Without compression, files are on average 2.27 times larger than the benchmark, indicating the high efficiency of the lossless deflate and shuffle filters. Relative to the DEFLATE benchmark, the lossy methods had average compression ratios of 0.51~~ A shows the distribution of compression ratios, normalized errors and timing statistics for the different methods. In the case of the compression ratios and normalized errors, results are presented separately for the dense and sparse variables. For dense variables, the median compression ratios were 1.3 (DEFLATE), 3.2 (NSD2), ~~0.66~~ 2.4 (NSD3), ~~0.77~~ 2.0 (NSD4), ~~0.91~~ 1.6 (NSD5), ~~0.35 (LSSCALAR) and 0.66 (LSARRAY). In three data sets (1, 3, 4)the~~ 4.2 (LIN) and 2.6 (LAY). For sparse variables, the median compression ratios were 2.0 (DEFLATE), 4.3 (NSD2), 3.3 (NSD3), 2.8 (NSD4~~and LSARRAY (shown in cyan and gray, respectively) produce very similar file sizes (within 4of one another). For the other three data sets LSARRAY yields file sizes between 71~~), 2.3 (NSD5), 7.4 (LIN) and ~~77of NSD4, roughly in between the NSD2 and~~ 5.2 (LAY). It can be seen that LIN gave the greatest compression, and the LAY compression ratios were comparable with those of NSD2 or NSD3~~file-sizes for these data sets. In each case, LSSCALAR produces the greatest compression~~ .

The median compression times (Figure 1~~presents the time taken by each compression method, and the time for extraction by LSARRAY (shown in white) . Averaged across the cases, the median standardized~~ B) normalized relative to the DEFLATE compression time were 0.82 (NSD2), 0.91 (NSD3), 0.79 (NSD4), 0.91 (NSD5), 0.57 (LIN), 3.45 (LAY compression) and 1.84 (LAY extraction). Differences between the bit-grooming methods were relatively small and slightly faster than DEFLATE alone, whereas the LIN compression was nearly twice as fast as DEFLATE. These values are consistent with DEFLATE compressing twice as much data for bit-grooming as for LIN, which store four and two bytes per value, respectively. The LAY times (both compression and decompression) were significantly slower than for the other methods, particularly for compression; this is most likely due to differences in implementation, as this LAY was programmed in Python while the

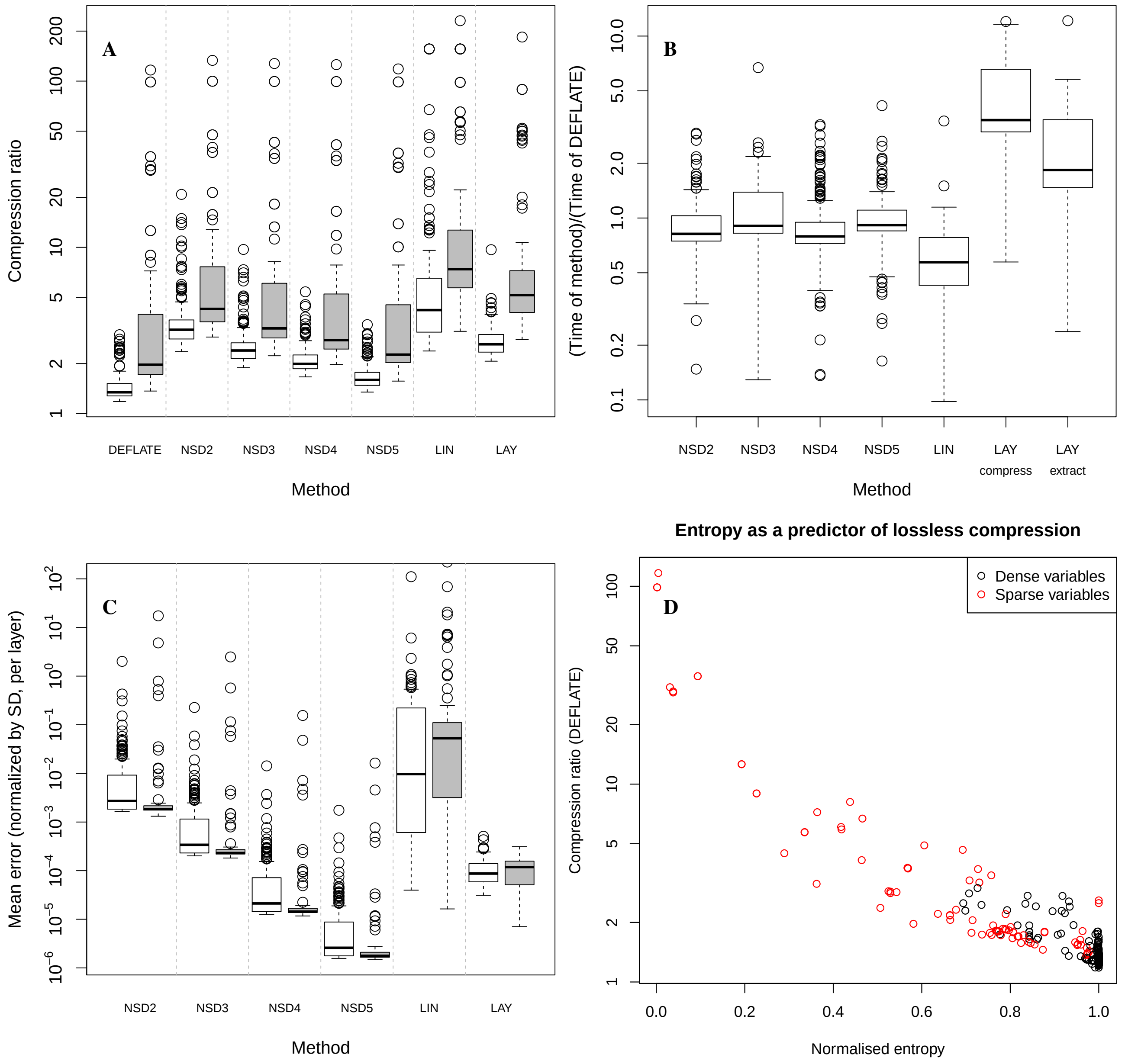

Figure 1. **A**: Distribution of compression ratios (original file size divided by compressed file size) measured for seven methods, applied to six test datasets (higher is better), plotted separately by dense variables and sparse variables (white and grey boxes, respectively). The box-plots (in panels A, B and C) were defined as follows: the thick line and the center of the box denotes the median, the bottom and top of the box show $q_{0.25}$ and $q_{0.75}$ (respectively) or the 0.25 and 0.75 quantiles of the distribution, the whiskers extend from $q_{0.25} - 1.5 \cdot \mathrm{IQR}$ to $q_{0.25} + 1.5 \cdot \mathrm{IQR}$, where IQR is the inter-quartile range ($q_{0.75} - q_{0.25}$) and the points shown are all outliers beyond this range. **B**: Distribution of scaled compression/decompression times for each method (lower is better), with LAY represented twice (for compression and extraction). These times are normalized by the compression time from DEFLATE. **C**: Distribution of errors, normalized by the per-layer standard deviation. **D**: The achieved lossless compression ratios (i.e. from DEFLATE) as a function of the normalized entropy (1.0 corresponding to the maximum theoretical for the data).

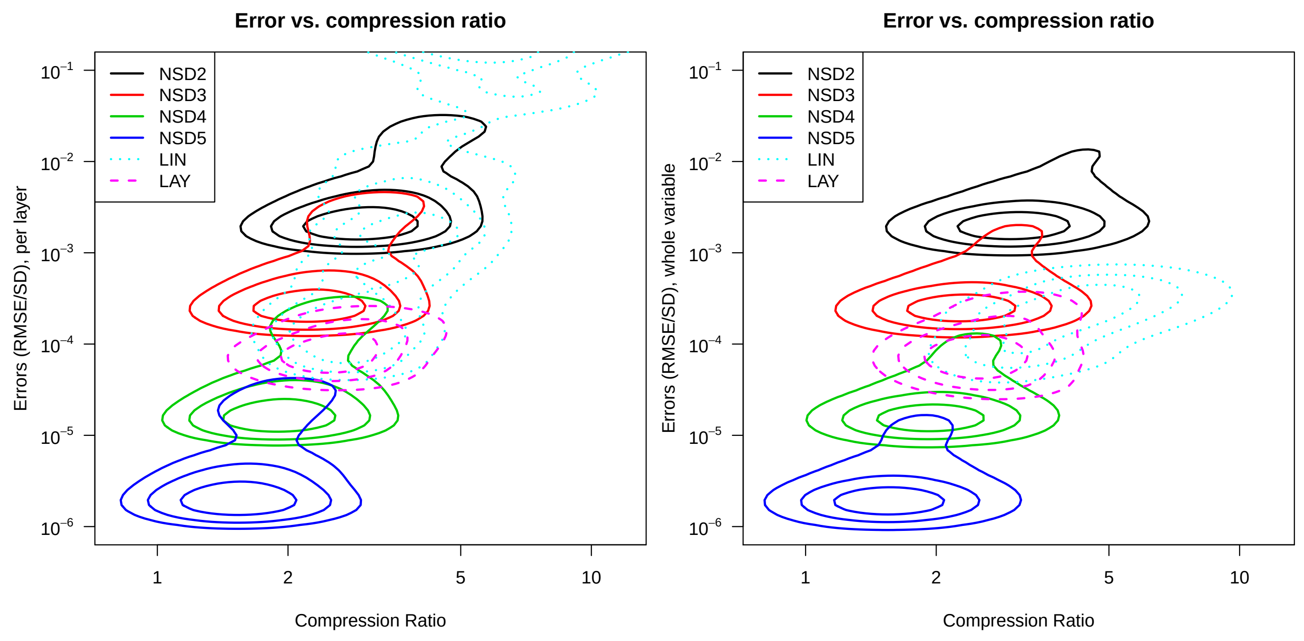

Figure 2. The relationship between normalized errors and compression ratio for the lossy compression methods considered. The three contours for each method show the bounds within which the two-dimensional kernel-smoothed distribution integrates to 0.25, 0.5 and 0.75, respectively. Only dense variables were used to produce this plot.

other methods used compiled C/C++ utilities. We believe that the overhead from loading some of the python libraries used in the implementation of LAY may cause this method to be relatively slower for smaller files; we note that most of the variables considered are relatively small, with uncompressed file sizes ranging from 1.9 MB to 65.6 MB. This hypothesis was supported by a test case where the suite of compression methods were applied to a much larger array[2] of size 1.5GB, the compression times were ~~0.37 (UNCOMPRESSED), 0.91~~ 103 s (DEFLATE), 89 s (NSD2), ~~1.00~~ 107 s (NSD3), ~~0.92~~ 97 s (NSD4), ~~1.04~~ 109 s (NSD5), ~~0.74 (LSSCALAR) and 0.88 (LSARRAY), with the time from DEFLATE serving as a benchmark within each dataset. The median standardized extraction time for LSARRAY was 1.39 (using the same normalization factor)~~69 s (LIN) and 70 s (LAY), while the unpacking time for LAY was 133 s.

~~Lossy~~ For all methods considered except DEFLATE, the compression comes at ~~a cost to the precision of~~ the ~~underlying data. Figure ?? summarizes the standardized errors . A value of 0.01 in this figure indicates that, a mean error ratio of 0.01 across thin slices of a given variable, where the error ratio is the root-mean squared error within the thin slice divided~~ expense of precision; the distribution of resultant errors is shown in Figure 1C (and Figure 1 of the online supplementary section). For

[2]The ERA-Interim (Dee et al., 2011) east-west wind component at 241 latitude, 480 longitudes, 60 vertical levels and 124 times, spanning 2015-07-01 00:00 UTC to 2015-07-31 18:00 UTC at 6-hourly intervals, converted from its original GRIB format. This dataset can be accessed through the ECMWF's public dataset portal (http://apps.ecmwf.int/datasets/)

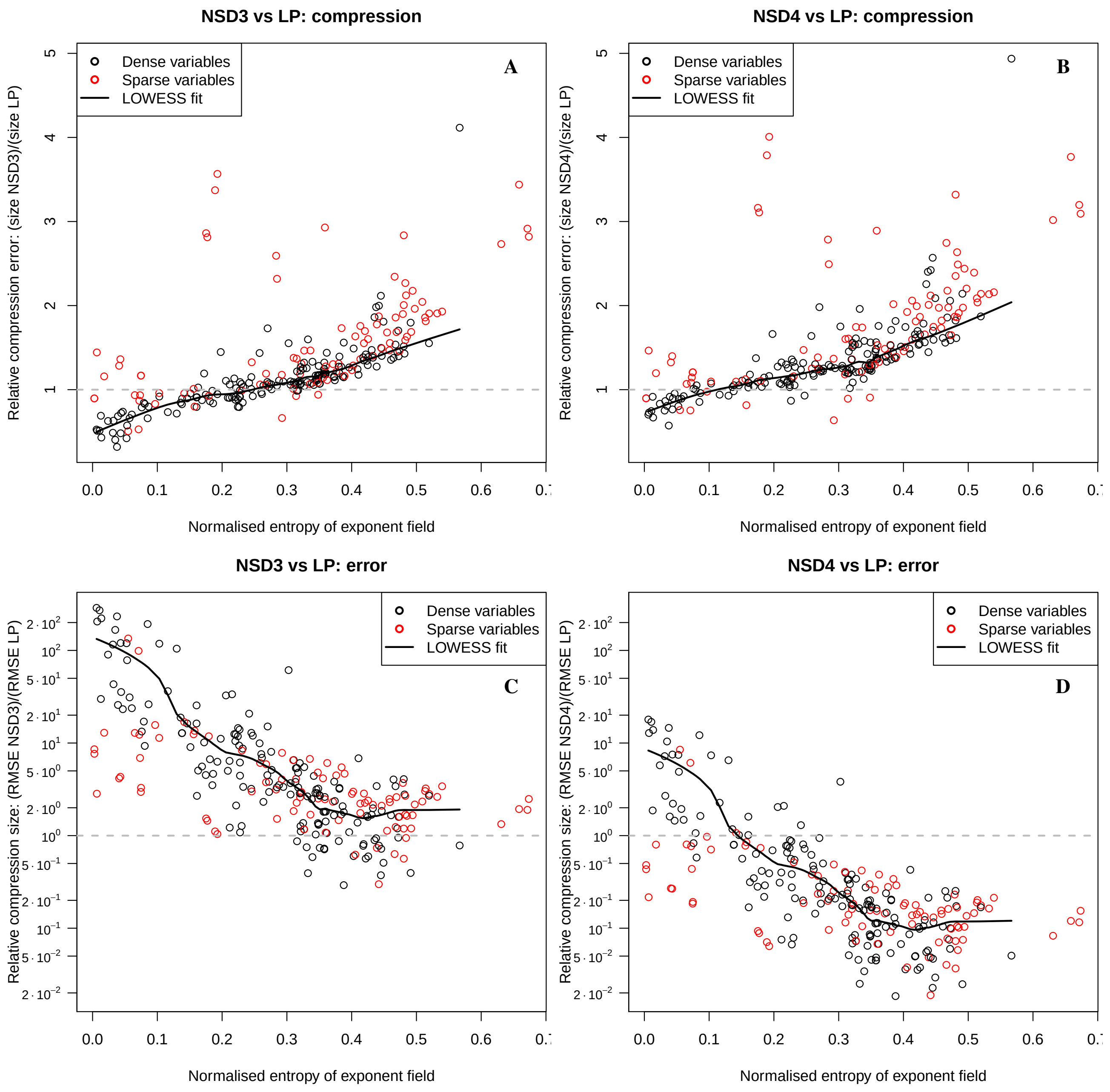

Figure 3. Relative performance of LAY compared with NSD3 (left column) or NSD4 (right column), in terms of compression (top row) and errors (bottom row). The grey dashed line indicates values of 1.0 on the y-axis. The LOWESS fit was based on the dense variables alone.

dense variables, the median relative errors shown in Figure 1C are $1.8 \cdot 10^{-3}$ (NSD2), $2.3 \cdot 10^{-4}$ (NSD3), $1.4 \cdot 10^{-5}$ (NSD4), $1.8 \cdot 10^{-6}$ (NSD5), $5.3 \cdot 10^{-2}$ (LIN), $1.2 \cdot 10^{-4}$ (LAY). Unsurprisingly, with each additional significant digit of precision requested of bit-grooming, the normalized errors fall by a factor of 10. The errors of LAY are comparable to those of NSD3 or NSD4 while for the metric shown in Figure 1C (RMSE/SD, calculated separately per thin slice, then averaging the ratios), LIN displays much larger errors than the other methods – the errors for LIN are, more than for the other lossy compression methods in this comparison, sensitive to the error metric used and this is discussed below. Dense variables show some differences in the distribution of errors compared to sparse variables; errors normalized by the standard deviation ~~(or mean) of the original values within the thin slice~~appear smaller and the errors normalized by the mean appear larger. This is because many of the sparse variables are zero at most points, and in many cases this tends to reduce the mean and increase the standard deviation (compared to examining only the non-zero values).

The choice of error metric ~~will highlight different aspects of the performance of each method. When normalizing the errors by the standard deviation of the original values, the~~ is ambiguous and leads to slightly different different results. Four different normalization factors for the RMSE were considered (shown in Figure 1 of the online supplementary section). It can be seen that the comments about the bit-grooming ~~methods appear to perform relatively poorly for fields where the baseline value is large relative to the within-slice standard deviation (e.g. temperatures stored with units of K, concentrations of well-mixed atmospheric trace gases such as CO_2 or CH_4) . In these cases, the first few significant figures may be constant across the slice and all variation is recorded in the last few significant figures. Arguably such fields should be stored with~~ and LAY methods in the preceding paragraph hold regardless of the normalization method, whereas LIN shows much higher standarized errors if errors are normalized within thin-slices and then averaged (due to the reasons explained in the last part of Section 2.2). Bit-grooming keeps precision loss to known bounds for each individual datum, LAY leads to roughly constant errors within thin-slices, and LIN results in roughly constant errors for the whole variable. This last point is illustrated in Figure 4 (especially panels A, B, C and E) of the Supplementary Material, which shows horizontal profiles of the (non-normalized) RMSE for a sample of six variables from among the 255 considered; Figure 5 in the Supplementary Material shows the corresponding relative errors.

The pairwise relationship between compression and error is shown in Figure 2, with the distribution shown based only on dense variables; the error metric used is the RMSE normalized by the standard deviation for the whole variable (rather than calculated normalizing separately per thin slice), which tends to mask large relative errors in certain sections of the data array for LIN. We see that the bit-grooming and LAY methods form something of a continuum, with LAY falling between NSD3 and NSD4. The linear slope on a log-log plot is suggestive of a ~~larger number of significant digits, however for simplicity with have used the same number of significant digits for all variables~~power-law relationship, which would be consistent with fundamental constraints on compression potential consistent with rate-distortion theory (Berger, 2003) .

~~When normalizing the errors by the mean of the original values (or in this case, by the absolute value of the mean of the original values), the layer-packing will appear to perform poorly for fields where the variation is large relative to~~ The question of when LAY is preferable to NSD3 or NSD4 can be addressed with reference to the complexity statistics. Among those complexity metrics considered, the ~~baseline value (e.g.~~ normalized entropy of the data field proved to be the best predictor of compression in the lossless case (DEFLATE). By contrast, the best predictor of the relative performance of LAY to the two bit-grooming methods was the normalized entropy of the exponent field (NEEF). By “best predictor”, we mean that these were, respectively, the most strongly correlated among the metrics considered with the DEFLATE compression ratios and the relative error or compressed file size. In the case of lossless compression, the correlation of the log of the compression ratio with the normalized entropy was over 0.9 (the next highest correlation was below 0.8, all variables were included), while for differentiating bit-grooming and LAY, the absolute correlations between the NEEF and the log of the file size ratio or the log of the RMSE ratio as shown in Figure 3 were both over 0.8 (c.f. ~~mass of snow, concentrations of short-lived atmospheric trace gases with few emission sources). In light of such considerations, the distributions of errors are presented for both normalization methods (figures **??** and **??**~~the next highest correlations were around 0.6, only dense variables were included).

~~The median standardized errors (across all variables and all datasets, normalised by the standard deviation) for the lossy methods were $2.09 \cdot 10^{-2}$ (NSD2), $2.56 \cdot 10^{-3}$ (NSD3), $1.77 \cdot 10^{-5}$ (~~The trade-off between error and compression is evident: as the NEEF increases, the bit-groomed file-sizes become larger than the corresponding LAY file-sizes, while the errors of resulting from LAY grow relative to those of bit-grooming (Figure 3). The LAY file sizes were larger than those of NSD3 or NSD4 ~~), $2.14 \cdot 10^{-6}$ (NSD5), $2.84 \cdot 10^{-2}$ (LSSCALAR)~~when the NEEF greater than 0.25 or 0.1, respectively (Figure 3, upper row). Errors of LAY were generally less than those of NSD3, while the errors for NSD4 were smaller for values of the NEEF greater than 0.15 (Figure 3, lower row). In the case when the normalised entropy of the exponent field is greater than 0.25, LAY yields both smaller files and lower errors than NSD3, while for all other cases the choice between LAY and NSD3 and ~~$9.48 \cdot 10^{-5}$ (LSARRAY). When normalised by the mean, the median standardized errors remained similar for the~~ NSD4 means deciding between smaller file sizes or smaller errors.

The reason that the NEEF differentiates the relative performance between bit-grooming ~~methods, and approximately doubled for the other methods (to $5.55 \cdot 10^{-2}$ for LSSCALAR and $1.80 \cdot 10^{-4}$ for LSARRAY). One clear result is that LSSCALAR caused a dramatic loss of accuracy in many cases, much more than one might expect given the nominal precision of roughly $1/65535 \approx 1.53 \cdot 10^{-5}$ times the range.~~ and LAY can be understood by the nature of the errors induced by the two techniques. Bit-grooming guarantees constant *relative* errors for each individual datum, whereas LAY results in errors that are roughly constant in *absolute* magnitude across a thin-slice. Assuming that the variable is dense, if the data within each thin-slice vary little (corresponding to a low NEEF), then LAY will achieve relatively small errors and since the compressed field contains more information, the resultant file size will be larger.

One interesting aspect of these results is that the entropy is defined *independent* of the values in the distribution (Figure 1), based solely on the frequencies of values in the array, yet the NEEF proved to be more informative regarding relative performance than other metrics that accounted for range in magnitude (e.g. the range or standard deviation of the exponent field, the logarithm of the largest non-zero value divided by the smallest non-zero value). This is likely due to the fact that the RMSE summarises the error distribution, weighting both by size and frequency, and the entropy is a measure of statistical dispersion.

Given the clear relationship between the normalized entropy of the data field (NEDF) and the compression ratios achieved by DEFLATE alone (figure 1D), it leads to the question of whether the NEDF is predictive of compression ratios achieved by the lossy methods. The NEDF is indeed highly predictive of the compression ratio of the reduced-precision fields as well

(Figure 2 of the supplementary section), with the absolute correlation exceeding 0.85 in each case. This may be expected, given the already clear relationship between these two parameters. Furthermore, the reduction in the NEDF is seen to be predictive of the "compression improvement", which we define as the DEFLATE-compressed file-size divided by the file-size achieved by the lossy compression method; this relationship is most apparent for those methods with the highest compression ratios, namely LIN, LAY and NSD2 (see Figure 3 of the supplementary section). The linear relationship, however, offers only a partial explanation (the correlation is over 0.8 for LIN and LAY, and over 0.6 for NSD2 and NSD3), however it appears to explain most of the variation when the lossy method achieves two-fold or greater compression relative to that obtained by DEFLATE.

~~The NSD5 method was consistently the most accurate, followed by NSD4, LSARRAY, NSD3, NSD2 and LSSCALAR. The NSD3, NSD4, NSD5 and LSARRAY methods were each accurate to 0.05or better (of the standard deviation or mean). In many applications this is much smaller than the uncertainty of the modelled or observed data. The trade-off between error and compression is illustrated in figure **??**~~

3 Discussion

Layer-packing was compared with scalar linear packing, bit-grooming and lossless compression via the DEFLATE algorithm. The lossy methods ~~show something of a continuous gradient when considering both these aspects. In terms of the errors, the LSARRAY method falls in between NSD3 and NSD4, with compression ratios between NSD2 and NSD3 for half of these datasets and close to NSD4 for the remaining files~~form a continuum when one compares the resultant compression ratios and normalized errors. The trade-off between error and compression has been shown elsewhere (e.g. Berger, 2003) . Despite the fact that LAY and LIN represented the data as two-byte integer arrays, which occupy half the storage of four-byte floating point numbers (ignoring the relatively minor contribution to LAY from the much smaller accompanying scale and offset arrays), it can be seen that both methods effectively fit into the continuum spanned by bit-grooming, which represents data as four-byte floats (Figure 2).

~~The data-points in figure **??** appear at first glance roughly linear on the log-log scales. Using linear regression models, we found that the intercepts were significantly different ($p \approx 10^{-5}$) , but that there was no significant difference between the slopes ($p \approx 0.4$) ; these comparisons were based on analysis of variance (ANOVA)F-statistics. In these models one outlier data-point was excluded (LSSCALAR for dataset 4)~~ In this study, we have effectively separated the precision-reduction from the compression itself. This is because the bit-grooming, LAY and LIN methods can be thought of as preconditioners for the same lossless compression algorithm (namely DEFLATE). The reduction in entropy due to these preconditioners explains a large part of the improved compression above lossless compression. This concept could be extended to develop methods for automatically determining the right precision to retain in a dataset.

This study did not extend to the comparison with other lossless filters for compression of the precision-reduced fields, nor did it compare the results with other lossy compression techniques. While it is possible that the findings presented here extend beyond the deflate and shuffle technique, other lossy and non-lossy compression algorithms operate in fundamentally different ways. Such an extension to this study may be considered in future, for example, by taking advantage of the fact that the HDF5

API allows for a range of alternative compression filters to be loaded dynamically (HDF5 Group, 2016) , to provide faster or more efficient compression than the default algorithms.

A number of such compression filters (both lossy and lossless) have been developed specifically for geophysical datasets, which have different properties compared to plain text, for example. They tend to be multi-dimensional, stored as floating-point numbers and in many cases are relatively smooth (Hübbe et al., 2013) . Specialised compression algorithms exist for such datasets that rely on the smoothness properties to confer a certain degree of predictability, which reduces the number of effective degrees of freedom in the dataset. For example, Hübbe et al. (2013) present a lossless algorithm termed MAFISC (Multidimensional Adaptive Filtering Improved Scientific data Compression) that incorporates a series of filters (some of which are adaptive), which in cases present gave stronger compression than the other lossless algorithms under consideration. In a similar spirit, Di and Cappello (2016) use a lossy, adaptive, curve-fitting technique that gives highly competitive compression performance while simultaneously constraining relative and/or absolute errors (as defined by the user).

Other authors have shown impressive data-compression rates using methods originally developed for image processing. For example, the GRIB2 format allows for compression using the JPEG-2000 algorithm and format (based on wavelet transforms) to store numerical fields (Skodras et al., 2001) . Woodring et al. (2011) describe a work-flow based around JPEG-2000 compression in order to ~~satisfy the assumptions of the linear models. The linear slope on a log-log plot is suggestive of a power-law relationship, which would be consistent with fundamental constraints on compression potential consistent with rate-distortion theory (Berger, 2003)~~ overcome bandwidth-limited connections while quantifying the ensuing reduction in precision. They demonstrate the same compression-error trade-off as illustrated in this work. Robinson et al. (2016) compare lossy compression via JPEG-2000, PNG (another graphics standard) and three video codecs; the video codecs showed very high relative errors (in the order of 0.1 to 1.0), but also very high compression rates (around 300-fold compression).

A number of studies have assessed a range of compression methods on a variety of datasets (Baker et al., 2014; Di and Cappello, 2016, e.
They show, amongst other things, that no one method provides the most effective compression for all datasets considered. Also apparent is that lossy methods tend to result in higher compression ratios than lossless techniques, and that methods designed specifically for scientific data (e.g. exploiting smoothness when it is present) are often highly competitive.

~~Left panel: Compression ratios (compressed file size divided by original file size) measured for six methods, applied to six test datasets (lower is better). The DEFLATE method serves as the benchmark and all values are equal to 1.0 by definition. Right panel: Scaled compression/decompression times for each method (lower is better), with LSARRAY represented twice (for compression and extraction). These times are normalized by the compression time from DEFLATE. The median values (shown at the right of each panel) are calculated across all datasets.~~

~~The distribution of standardized errors (lower is better) of the four lossy compression methods, applied to six test datasets. Each data-point behind the box-whisker plot corresponds to one variable in a given dataset, based on the mean standardized error per variable normalized by the standard deviation (left panel) or the mean absolute value (right panel) of the original data. Whiskers indicate the minimum and maximum values in the sample, box edges correspond to the 25and 75quantiles, and the center-line indicates the sample median. Methods are shaded with the same color as in Figure 1. The median values (shown at the right of each panel) are calculated across all variables within these datasets.~~

~~The median error per file (from figure ??) plotted against the corresponding compression ratio for that file (from the left panel of figure 1). Errors are normalized by the standard deviation (left panel) or the mean (right panel) of the original values within each thin slice.~~

4 ~~Discussion~~

~~This paper introduces~~

Layer-packing achieves compression and error results roughly in between bit-grooming storing three or four significant figures. Layer-packing and bit-grooming control error in different ways, with bit-grooming guaranteeing fixed relative errors for every individual datum while layer-packing ~~as a variant of the linear scaling technique for data compression , and compares the results to this and other lossless and lossy compression techniques.~~ results in roughly constant relative errors within the hyper-slice across which the packing is applied.

The idea itself behind layer-packing is not new, and forms the basis of compression within the GRIB format, in ~~that~~ which each field is a two-dimensional array ~~,~~ and compression is performed on fields individually. ~~Although the methods described are used only on netCDF data files, they apply equally to other data formats (e.g.HDF4, HDF5). These results are presented as a proof-of-concept only.~~

~~The comparison with GRIB (especially GRIB2, which compresses via JPEG-2000, based on wavelet approximation) is especially relevant. The procedure described here~~ In one sense layer-packing is more general than that used in GRIB, in that thin-slices are not restricted to being two-dimensional. Our preliminary results (not shown) showed that the JPEG2000 algorithm yields greater compression compared to the methods presented here for the same level of error; this echoes the findings of Caron (2014) , which describe the efficient compression achieved. However like scalar linear packing, JPEG2000 does not offer clear controls about the resultant errors and thus some experimentation (in setting the number of bits per value) is needed to avoid excessive loss of precision. More limiting, perhaps, are the technical procedures required to convert generic netCDF data into GRIB2 format, including meeting constraints on variable and dimension names. The GRIB format ~~is rather restrictive in terms of~~ only allows storing meta-data that match predefined tables, and is thus nowhere near as general or self-describing as HDF5 (or its derivatives such as netCDF-4). ~~On the other hand GRIB2 is very space efficient; Caron (2014) estimated that GRIB2 files are on average 44of the size of the equivalent deflate-compressed netCDF-4 files (n.b~~This, combined with the software requirements (e.g. the JPEG library) beyond netCDF for decompression render GRIB-compression unattractive for general purpose usage. Outside of organisations with strong technical support for modelling operations (e.g. ~~relative errors were not reported, which limits the comparison)~~operational weather prediction centres), its suitability may be limited to special purposes and expert users.

~~The deflate/shuffle algorithms we tested are the only two lossless filters accessible through both the HDF5 and netCDF-4 APIs, which makes their performance of the greatest interest and widest applicability.We also note that the~~

Although the methods described here are used only on netCDF data files, they apply equally to other data formats (e.g. HDF4, HDF5~~format allows for other filters to be used for lossless compression, not just the deflate/shuffle algorithms used~~

~~here. A range of other lossless compression filters can be loaded dynamically (HDF5 Group, 2016) , many of which offer faster or more efficient compression than the default algorithms (Hübbe and Kunkel, 2013, e.g. as discussed by) . The compression performance of the different lossy (packing or bit-grooming)filters when combined with these other lossless filters was not explored within this study but is likely to be similar to that observed for the deflate/shuffle algorithms applied here.~~). These results are presented as a proof-of-concept only.

~~Both~~ The timing information is presented mainly for completeness and a caveat should be raised. As noted above, the ~~linear scaling and the layer packing~~ compression via DEFLATE, NSD2, NSD3, NSD4, NSD5 and LIN were performed using tools from the NCO bundle (written in C and C++), whereas the LAY compression was implemented in Python. The code is presented as a demonstration of layer-packing rather than production code.

Both the scalar linear packing and the layer-packing use the same representation of the data (i.e. two-byte integers), however large differences in the compression and relative errors were found. This is because the compressibility of a packed field is related to the distribution of values within the scaling range. ~~This can be illustrated by the following example. Consider a three-dimensional field that ranges over three orders of magnitude in the vertical dimension, taking values of $O(10^3)$ at bottom and values of $O(10^0)$ at the top. Suppose the range across the entire field is 0.0 to 5000.0 (ignoring units), and thus the smallest increment when recorded at single precision (with linear scaling) is roughly $5000.0/65536 \approx 0.08$ across the entire field. The upper levels will be much more heavily quantized, relative to their values , compared to the lower levels, and thus will be subject to both higher compression and higher relative errors. Compressing the same field using layer packing may yield, for the sake of example, a range of 2000.0 at the bottom level and 5.0 in the top level, and in both cases the precision will be greater, and hence more values will be taken within each vertical level. As such, the more variable the field, the less compression will be obtained from the combination of the deflate and shuffle filters. Hence linear scaling gives a lower precision, less spatial variation within the thin slice and hence is more compressible. Another factor that makes layer packing less compressible is that the accompanying scale-offset parameter arrays must be stored.~~

~~Using the lossless deflate and shuffle algorithms, the compressed files were less than 50of the original uncompressed size. Beyond this, the lossy compression methods achieved further space savings, although to varying degrees for the individual datasets. There was a general trade-off between precision-loss and compression efficiency (figure **??**). When considered along both these axes, layer-packing has errors between those of NSD3 and NSD4. For half the datasets considered (1, 3, and 4) it yielded compressed sizes in between those of NSD2 and NSD3 and in the other half it yielded file sizes comparable to those of NSD4. In other words, in three of the cases considered~~ Across a given thin slice, layer-packing will represent values using the full range of two-byte integers, whereas scalar linear packing will typically use a smaller portion of that range. This increases the loss of precision resulting from layer-packing ~~gave a worse compression-error trade-off (with file-size close to NSD4 but larger errors) while in remaining three cases it performed better (with both smaller errors and smaller file size than NSD3)~~but also entails greater compression.

~~Two error metrics were employed. Each emphasized different aspects of the performance. When normalizing by the mean (of the entire variable, or of the thin-slice), variables with a high standard deviation to mean ratio will show larger errors using the layer-packing compression. Whereas when normalizing by the standard deviation, variables with a high mean to standard~~

~~deviation ratio will show larger errors using the bit-grooming compression. These considerations highlight the importance of understanding the properties of lossy filters when selecting the most appropriate filter for a given application.~~

When considering which compression method to use for an individual dataset, one needs to consider several factors. First, space constraints and the size of the datasets in question will vary considerably for different applications. Second, the degree of precision required will also depend on the application, and may differ between variables within a dataset. The bit-grooming (storing at least three significant digits) and layer-packing techniques achieved average normalized errors of 0.05% or better, which in many geoscientific applications is much less than the model or measurement errors. Third, datasets vary considerably in their inherent compressibility~~.~~ , which in the cases considered appeared strongly related to the normalized entropy of the data array. Fourth, how data are stored should also reflect how it will be used (e.g. active use versus archiving). A major disadvantage of the layer-packing as described here is that it is essentially an archive format, and needs to be unpacked by a custom application before it can be easily interpreted. Scalar linear packing is similarly dependent on unpacking (although many netCDF readers will automatically unpack such data, from two-byte integers to four-byte floating point, by default), whereas bit-grooming requires no additional software. Finally, other issues relating to portability, the availability of libraries and consistency within a community also play a role in determining the most appropriate storage format.

4 Conclusions

This paper considers ~~different forms of linear-scaling~~ layer-packing, scalar linear packing and bit-grooming as a basis for compressing large gridded datasets. ~~Layer-packing, scalar linear packing and bit-grooming were compared. Layer-packing was found to be a competitive format for archive purposes, with benefits (~~When viewed in terms of the compression-error trade-off~~) for some datasets. In most cases, it was comparable to storing between 3 and 4 significant digitsper datum~~, layer-packing was found to fit within the continuum of bit-grooming (i.e. when varying the number of significant digits to store), roughly in between storing three and four significant digits. The relative performance of layer-packing and bit-grooming was strongly related to the normalized entropy of the exponent array, and again highlighted the trade-off between compression and errors. Given the variation in compression and accuracy achieved for the different datasets ~~considere~~considered, the results highlight the importance of testing compression methods on a realistic sample of the data.

If space is limited and a large dataset must be stored, then we recommend that the standard deflate and shuffle methods be applied. If this does not save enough space, then careful thought should be given to precisely which variables and which subsets of individual variables will be required in future; it often arises that despite a wealth of model output, only a limited portion will ever be examined. Many tools exist for sub-setting such datasets. Beyond this, if further savings are required and if the data need not be stored in full precision, then the appropriate relative precision for each variable should be selected and applied via bit-grooming~~methods should be trialled~~. Layer-packing should be ~~reserved for archive applications, and should be compared with bit-grooming.~~ considered when choosing a compression technique for specialist archive applications.

5 Code availability

The python-based command line utilities used for the layer-packing, unpacking and relative-error analysis are available freely at https://github.com/JeremySilver/layerpack.

6 Data availability

Of the datasets used in the tests (listed in ~~section~~Section 2.1.1), datasets ~~2-6~~ 2–6 are available online at https://figshare.com/projects/Layer_Packing_Tests/14480 along with a brief description of each file. Dataset 1 could not be distributed with the other files due to licensing restrictions but can be accessed through the ECMWF's public dataset portal (http://apps.ecmwf.int/datasets/), using the following set of inputs: stream = synoptic monthly means, vertical levels = pressure levels (all 37 layers), parameters = all 14 variables, dataset = interim_mnth, step = 0, version = 1, time = 00:00:00, 06:00:00, 12:00:00, 18:00:00, date = 20080901 to 20081201, grid = $1.5^\circ \times 1.5^\circ$, type = analysis, class = ERA Interim.

Author contributions. J. D. Silver wrote the layer-packing python software, performed the compression experiments and wrote most of the manuscript. C. S. Zender contributed to the design of the study, provided some of the test datasets used in the experiments and contributed to the text.

7 Competing interests

The authors declare that they have no conflict of interest.

Acknowledgements. The work of J. D. Silver was funded by the University of Melbourne's McKenzie Postdoctoral Fellowship programme. The work of C. S. Zender was funded by NASA ACCESS NNX12AF48A and NNX14AH55A and by DOE ACME DE-SC0012998. We thank Peter J. Rayner (University of Melbourne) for useful discussions.

References

Baker, A. H., Xu, H., Dennis, J. M., Levy, M. N., Nychka, D., Mickelson, S. A., Edwards, J., Vertenstein, M., and Wegener, A. (2014). A methodology for evaluating the impact of data compression on climate simulation data. In *Proceedings of the 23rd International Symposium on High-performance Parallel and Distributed Computing*, pages 203–214. ACM.

Berger, T. (2003). Rate-distortion theory. In *Wiley Encyclopedia of Telecommunications*. John Wiley & Sons, Inc.

Brasseur, G., Hauglustaine, D., Walters, S., Rasch, P., Müller, J.-F., Granier, C., and Tie, X. (1998). MOZART, a global chemical transport model for ozone and related chemical tracers: 1. Model description. *Journal of Geophysical Research: Atmospheres*, 103(D21):28265–28289.

Caron, J. (2014). Converting GRIB to netCDF-4: Compression studies. www.ecmwf.int/sites/default/files/elibrary/2014/13711-converting-grib-netcdf-4.pdf, Last accessed: 2016-06-17. Presentation to the workshop “Closing the GRIB/netCDF gap”, held at European Centre for Medium Range Weather Forecasts (ECMWF) at Reading, UK, 24-25 September 2014.

Dee, D., Uppala, S., Simmons, A., Berrisford, P., Poli, P., Kobayashi, S., Andrae, U., Balmaseda, M., Balsamo, G., Bauer, P., and Bechtold, P. (2011). The ERA-Interim reanalysis: Configuration and performance of the data assimilation system. *Quarterly Journal of the Royal Meteorological Society*, 137(656):553–597.

Dennis, J. M., Edwards, J., Evans, K. J., Guba, O., Lauritzen, P. H., Mirin, A. A., St-Cyr, A., Taylor, M. A., and Worley, P. H. (2012). CAM-SE: A scalable spectral element dynamical core for the Community Atmosphere Model. *Int. J. High Perform. Comput. Appl.*, 26:74–89.

Deutsch, L. P. (2008). DEFLATE compressed data format specification version 1.3. Technical Report Tech. Rep. IETF RFC1951, Internet Engineering Task Force.

Di, S. and Cappello, F. (2016). Fast error-bounded lossy HPC data compression with SZ. *Proc. IPDPS. IEEE.*

Goldberg, D. (1991). What every computer scientist should know about floating-point arithmetic. *ACM Computing Surveys (CSUR)*, 23(1):5–48.

HDF5 Group (2002). Performance evaluation report: gzip, bzip2 compression with and without shuffling algorithm. https://support.hdfgroup.org/HDF5/doc_resource/H5Shuffle_Perf.pdf.

HDF5 Group (2016). Filters. www.hdfgroup.org/services/filters.html, Last accessed: 2016-06-17.

Hübbe, N. and Kunkel, J. (2013). Reducing the HPC-datastorage footprint with MAFISC – Multidimensional Adaptive Filtering Improved Scientific data Compression. *Computer Science-Research and Development*, 28:231–239.

Hübbe, N., Wegener, A., Kunkel, J. M., Ling, Y., and Ludwig, T. (2013). Evaluating lossy compression on climate data. In *International Supercomputing Conference*, pages 343–356. Springer.

Huffman, D. (1952). A method for the construction of minimum-redundancy codes. *Proceedings of the IRE*, 40(9):1098–1101.

Rienecker, M. M., Suarez, M. J., Gelaro, R., Todling, R., Bacmeister, J., Liu, E., Bosilovich, M. G., Schubert, S. D., Takacs, L., Kim, G.-K., et al. (2011). MERRA: NASA’s modern-era retrospective analysis for research and applications. *Journal of Climate*, 24(14):3624–3648.

Robinson, N. H., Prudden, R., and Arribas, A. A. (2016). A practical approach to spatiotemporal data compression. *arXiv preprint arXiv:1604.03688*.

Skamarock, W. C., Klemp, J. B., Dudhia, J., Gill, D. O., Barker, D. M., Wang, W., and Powers, J. G. (2005). A description of the advanced research WRF version 2. Technical Report NCAR/TN-468 STR, National Center For Atmospheric Research, Boulder, Colarado, USA.

Skodras, A., Christopoulos, C., and Ebrahimi, T. (2001). The JPEG 2000 still image compression standard. *IEEE Signal processing magazine*, 18(5):36–58.

Unidata (2016). The netCDF User's Guide. Technical report, Unidata, UCAR, Boulder, CO, USA. www.unidata.ucar.edu/software/netcdf/docs/user_guide.html, Last accessed: 2016-06-17.

Woodring, J., Mniszewski, S., Brislawn, C., DeMarle, D., and Ahrens, J. (2011). Revisiting wavelet compression for large-scale climate data using JPEG 2000 and ensuring data precision. In *IEEE Symposium on Large Data Analysis and Visualization*, pages 31–38. IEEE.

Zender, C. S. (2008). Analysis of self-describing gridded geoscience data with netCDF Operators (NCO). *Environmental Modelling & Software*, 23(10):1338–1342.

Zender, C. S. (2016). Bit Grooming: Statistically accurate precision-preserving quantization with compression, evaluated in the netCDF Operators (NCO, v4.4.8+). *Geoscientific Model Development Discussions*, 2016:1–18.

Zender, C. S., Bian, H., and Newman, D. (2003). Mineral Dust Entrainment And Deposition (DEAD) model: Description and 1990s dust climatology. *J. Geophys. Res.*, 108(D14):4416.

Ziv, J. and Lempel, A. (1977). A universal algorithm for sequential data compression. *IEEE Trans. on Information Theory*, 23(3):337–343.

Ziv, J. and Lempel, A. (1978). Compression of individual sequences via variable-rate coding. *IEEE Trans. on Information Theory*, 24(5):530–536.

~~urlstyle~~

~~Berger, T. (2003), Rate-distortion theory, in *Wiley Encyclopedia of Telecommunications*, John Wiley Sons, Inc., .~~

~~Brasseur, G., D. Hauglustaine, S. Walters, P. Rasch, J.-F. Mller, C. Granier, and X. Tie (1998), MOZART, a global chemical transport model for ozone and related chemical tracers: 1. Model description, *Journal of Geophysical Research: Atmospheres*, *103*(D21), 28,265–28,289.~~

~~Caron, J. (2014), Converting GRIB to netCDF-4: Compression studies, , Last accessed: 2016-06-17, presentation to the workshop "Closing the GRIB/NetCDF gap", held at European Centre for Medium Range Weather Forecasts (ECMWF) at Reading, UK, 24-25 September 2014.~~

~~Dee, D., S. Uppala, A. Simmons, P. Berrisford, P. Poli, S. Kobayashi, U. Andrae, M. Balmaseda, G. Balsamo, P. Bauer, and P. Bechtold (2011), The ERA-Interim reanalysis: Configuration and performance of the data assimilation system, *Quarterly Journal of the Royal Meteorological Society*, , 553–597.~~

~~Dennis, J. M., J. Edwards, K. J. Evans, O. Guba, P. H. Lauritzen, A. A. Mirin, A. St-Cyr, M. A. Taylor, and P. H. Worley (2012), CAM-SE: A scalable spectral element dynamical core for the Community Atmosphere Model, *Int.J. High Perform.Comput.Appl.*, *26*, 74–89, .~~

~~Deutsch, L. P. (2008), DEFLATE compressed data format specification version 1.3, *Tech. Rep. Tech. Rep. IETF RFC1951*, Internet Engineering Task Force.~~

~~Group, H. (2016), Filters, , Last accessed: 2016-06-17.~~

~~Hbbe, N., and J. Kunkel (2013), Reducing the HPC-datastorage footprint with MAFISC – Multidimensional Adaptive Filtering Improved Scientific data Compression, *Computer Science-Research and Development*, *28*, 231–239.~~

~~Rienecker, M. M., M. J. Suarez, R. Gelaro, R. Todling, J. Bacmeister, E. Liu, M. G. Bosilovich, S. D. Schubert, L. Takacs, G.-K. Kim, et al. (2011), MERRA: NASA's modern-era retrospective analysis for research and applications, *Journal of Climate*, , 3624–3648.~~

~~Skamarock, W. C., J. B. Klemp, J. Dudhia, D. O. Gill, D. M. Barker, W. Wang, and J. G. Powers (2005), A description of the advanced research WRF version 2, *Tech. Rep. NCAR/TN-468 STR*, National Center For Atmospheric Research, Boulder, Colorado, USA.~~

~~Unidata (2016), The NetCDF User's Guide, *Tech. rep.*, Unidata, UCAR, Boulder, CO, USA, , Last accessed: 2016-06-17.~~

~~Zender, C. S. (2008), Analysis of self-describing gridded geoscience data with netCDF Operators (NCO), *Environmental Modelling Software*, , 1338–1342.~~

~~Zender, C. S. (2016), Bit Grooming: Statistically accurate precision-preserving quantization with compression, evaluated in the netCDF Operators (NCO, v4.4.8+), *Geoscientific Model Development Discussions*, *2016*, 1–18, .~~

~~Zender, C. S., H. Bian, and D. Newman (2003), Mineral Dust Entrainment And Deposition (DEAD) model: Description and 1990s dust climatology, *J. Geophys. Res.*, *108*(D14), 4416, .~~

~~Ziv, J., and A. Lempel (1977), A universal algorithm for sequential data compression, *IEEE Trans. on Information Theory*, , 337–343.~~

~~Ziv, J., and A. Lempel (1978), Compression of individual sequences via variable-rate coding, *IEEE Trans. on Information Theory*, , 530–536, .~~