bqbtool User Manual

A Lossless Compression Tool for Trajectories of Atom Positions and Volumetric Data

Martin Brehm & Martin Thomas, 2018

`https://brehm-research.de/bqb`

Manual Version: 01.10.2018

1. Introduction

The bqbtool is a command line utility which compresses simulation trajectories to the bqb file format. Both standard trajectories (*i. e.*, containing the position of the atoms) and volumetric data trajectories (*i. e.*, containing a grid of values, such as total electron density, in every frame) can be compressed. The compression is lossless, which means that the decompressed output will be bitwise identical to the original trajectory if the precision set by the user matches the input data precision. Our approach reaches a compression ratio of around 15:1 for position trajectories, and around 35:1 for volumetric trajectories. This is a much larger compression ratio than achieved with standard file compressors such as bzip2.

For details on the compression algorithm and performance, including some examples and benchmarks, please read (and cite) our article [1] which was published in September 2018:

M. Brehm, M. Thomas: "An Efficient Lossless Compression Algorithm for Trajectories of Atom Positions and Volumetric Data", *J. Chem. Inf. Model.* **2018**, *in press*,
DOI 10.1021/acs.jcim.8b00501

2. How to Obtain and Install

The homepage of the bqbtool and all other things related to the bqb format is

`https://brehm-research.de/bqb`

There, you can download the bqbtool and this manual for free. All parts of this project are licensed under the GNU LGPL v3 license, and are therefore free software. They can even be included into proprietary program packages (therefore „**L**GPL"), as long as the article **[1]** is cited, and the authorship is displayed in the program documentation and at runtime. For details, please see the LGPL license text. The bqbtool is written in C++ and currently contains around 37 000 lines of (freely available) source code. It is platform independent and does not require any external libraries.

Please note that the full bqbtool is integrated into the TRAVIS program package **[2,3]**. If you have a recent version of TRAVIS *(October 2018 or later)*, you don't need to intall the bqbtool at all. Just use the TRAVIS executable as if it was the bqbtool:

```
travis compress postraj in.xyz out.bqb
```

Apart from that, there are two ways how to obtain a working bqbtool:

- For systems running on Microsoft Windows, you can download one of the pre-compiled executable files from our homepage. We offer executables for 32 bit and 64 bit operating systems. We recommend to use the 64 bit version if possible, because it was built with a more recent compiler and therefore will run faster.

- For Linux/Unix operating systems you can download an archive file with the source code of bqbtool. Simply extract the archive, enter the directory, and type „`make`". As bqbtool does not require any external libraries, no setup and preparations should be required. You only require a working C++ compiler *(which should exist on every Unix system)*. To modify compiler options, edit the Makefile.

In both cases, you obtain a single executable file with the name „`bqbtool`". This is a command line utility – it does not possess a user interface. You pass options to the tool via command line arguments. For a short overview of supported command line options, simply call the executable without any further arguments.

We are also planning to publish a C++ library („libbqb") which can be used to equip any software package with support for reading and writing compressed bqb files. Then, simulation programs such as CP2k or LAMMPS might directly write compressed trajectories, while visualization programs such as VMD could directly read the compressed files without prior decompression. It will take a few more months until the libbqb is ready for its first public release *(hopefully end of 2018)*. Please stay tuned.

3. Basic Usage

With the current version of the bqbtool, you can [compress/decompress] [binary files/position trajectories/volumetric trajectories]. This makes up for exactly six different possible operations. More operating modes (such as splitting/merging trajectories, comparing files, checking file integrity) will be added in future releases. We guarantee that the bqb file format retains forward/backward compatibility; therefore, you will be able to work with your compressed bqb files with any future software version.

The general command line reads

```
bqbtool [compress/decompress] [postraj/voltraj/file] [options] [input] [output]
```

Make sure to put any options **before** the input/output file names.

3.1. Position Trajectories

To compress an existing position trajectory, simply enter

```
bqbtool compress postraj input.xyz output.bqb
```

Only XMol XYZ files [4] are currently supported as input trajectories. More formats will be added in future releases.

Please use unwrapped trajectories; wrapping leads to discontinuities which significantly reduce the compression efficiency *(see article* [1] *)*.

If you want to compress only the first 100 steps of the trajectory, specify

```
bqbtool compress postraj -steps 100 input.xyz output.bqb
```

The working precision is set to 10^{-5} Angstrom by default. To reduce the precision to 10^{-3} Angstrom (increases compression ratio), use

```
bqbtool compress postraj -prec 3 input.xyz output.bqb
```

The bqbtool optimizes the compression parameters in the beginning of each compression run to find the parameter set which yields maximal compression ratio. This can take some time. To switch off this automated parameter optimization, add „`-optimize 0`“ to the command line (before the input/output file names). To perform an even more exhaustive search for good parameters, try „`-optimize 3`“. The default value is „`-optimize 2`“ for position trajectories.

To decompress a bqb position trajectory, simply enter

```
bqbtool decompress postraj input.bqb output.xyz
```

Also here, only XMol XYZ files [4] are currently supported, and more formats will come.

To decompress only the first 100 steps, write in analogy to above

```
bqbtool decompress postraj -steps 100 input.bqb output.xyz
```

3.2. Volumetric Data Trajectories

A volumetric data trajectory is a sequence of uniform 3D grids containing real values. Typical examples are electron densities from quantum chemical calculations, often stored in Gaussian Cube file format [5]. Such volumetric trajectories are huge files, but are very useful, *e. g.*, for the computation of bulk phase vibrational spectra from MD simulations [6-8].

For volumetric trajectories, most instructions from the position trajectory part above still hold. To compress a volumetric trajectory in Gaussian Cube file format [5], enter

```
bqbtool compress voltraj input.cube output.bqb
```

If you want to compress only the first 100 steps, you can add „`-steps 100`“ to the command line, as explained above.

When compressing volumetric trajectories, the working precision is specified in relative rather than absolute accuracy, *i. e.*, number of significant digits. The default is to store 5 significant digits, as specified in the original Gaussian Cube format. To store 7 significant digits instead *(at the cost of lower compression ratio)*, add „`-vsigni 7`“. The maximum allowed number of significant digits is 9.

Often, volumetric trajectories also contain atom positions, which are stored together with the volumetric data. The absolute precision of the atom positions has a default value of 10^{-5} Angstrom. To set it to 10^{-3} Angstrom, add „`-pprec 3`“ to the command line.

As explained above, the bqbtool optimizes the compression parameters in the beginning of each compression run to find the parameter set which yields maximal compression ratio. This can take a lot of time for volumetric trajectories. To switch off this automated parameter optimization, add „`-optimize 0`“ to the command line. To perform an even more exhaustive search for good parameters, try „`-optimize 2`“ or even „`-optimize 3`“ (might take *days*...). The default value is „`-optimize 1`“ for volumetric trajectories.

To decompress a bqb volumetric trajectory, simply enter

```
bqbtool decompress voltraj input.bqb output.cube
```

Also here, you could specify „`-steps 100`“ to only decompress the first 100 steps.

3.3. Binary Files

You can also use the bqbtool to compress and decompress arbitrary binary files. The compression ratio will be very similar to bzip2, because most of the algorithms are identical. Therefore, there is no significant reason to use bqbtool instead of bzip2 for compressing binary files... This was just implemented because it came at no additional coding effort :-)

```
bqbtool compress file input.any output.bqb
```

```
bqbtool decompress file input.bqb output.any
```

4. Advanced Usage

4.1. Log File and Multiple Instances

The bqbtool writes a log file to „`bqbtool.log`“. All screen output is also found in the log file. If the file cannot be opened for writing, the program aborts execution right in the beginning *(be sure to have writing permission in the current directory)*. If already another instance of bqbtool writes to this log file, the file will be garbled. Therefore, please make sure to never run multiple instances of bqbtool at the same time in the same directory.

4.2. Re-using Compression Parameters

As described above, the bqbtool performs an automated tuning of compression parameters in the beginning of each compression run. This can take a considerable amount of time. If you want to compress many similar trajectories, it can be assumed that a good set of compression parameters will work for all of them, and it would be a waste of time to redo the parameter tuning in each run.

There are around 50 parameter which influence the compression ratio, and it would be tedious to specify all of them in the command line (still, this is possible). For a description of all these parameters, please see our article **[1]**. The bqbtool condenses all of those parameters into a short alphanumeric code, called a „parameter key“. Such a key is written to screen and log file after the automated tuning of parameters has finished, it contains all the optimized parameters in packed form:

```
Parameter key:  @1nzGImf2ekFo46
```

If you want to compress another trajectory with exactly the same parameter set, just specify the parameter key in the command line:

```
bqbtool compress postraj -pkey @1nzGImf2ekFo46 input.xyz output.bqb
```

Specification of a parameter key automatically disables the automated parameter tuning (as „`-optimize 0`“ would do), and therefore saves a considerable amount of time.

Parameter keys always start with a @ sign. Please note that keys from position trajectories and volumetric trajectories cannot be swapped.

If you simply want to expand a parameter key to human-readable values without compres-sing anything, you can use the „`-dryrun`“ command line option. It causes bqbtool to quit directly before the actual operation would start. The parameters are written to screen before. Then, also the output file can be omitted *(but an input file is still required)*:

```
bqbtool compress postraj -pkey @1nzGImf2ekFo46 -dryrun input.xyz
```

Command line arguments are handled in the order they have been specified. If you want to load a parameter key but want to change a few parameters, simply write the corresponding arguments behind the „`-pkey ...`“ part. You will obtain a new modified parameter key.

4.3. Truncating small Volumetric Data Elements

Volumetric data trajectories are often used to store the electron density *(from an ab initio molecular dynamics simulation)* along the trajectory. If a single molecule *(or a cluster of molecules)* in vacuum was simulated, there are regions inside the simulation box where the electron density is very small. One might expect that it decays to zero where no atoms are present. However, as the volumetric data is stored with a fixed number of significant digits, it will never become zero, but rather some very small values which essentially contain numerical noise. As noise can neither be compressed nor predicted by extrapolation, this significantly reduces the compression ratio. The bqbtool therefore offers to truncate all volumetric data elements to zero if their absolute value is below a given threshold. Per default, this threshold is set to 10^{-12}, *i. e.*, 10^{-12} is the smallest value above zero that can be represented. To set this threshold to 10^{-6} instead, add „`-veps 6`" to the command line. The smallest possible threshold is 10^{-63}, activated with „`-veps 63`".

Strictly speaking, the algorithm is no longer lossless if it truncates small values. However, these small values do not bear any physical meaning and are only numerical artifacts. Apart from that, such small values in electron density only occur in gas phase simulations with large vacuum regions. In bulk phase systems, the electron density never becomes so low at any point.

4.4. Random Access and Key Frames

Our compression algorithm uses temporal extrapolation techniques to reach high compression ratios. This requires the knowledge of a history of frames to decompress a given frame. On the other hand, the bqb file format allows fast random access to individual frames *(see section 5.3 below)*. You could, *e. g.*, easily obtain frame 1000 of a compressed trajectory; but to decompress it, you would require the decompressed frames 995 - 999. As each frame depends on its predecessors, you would have to start in frame 1 and decompress the whole trajectory, until you reach frame 1000. This is of course not what „random access" means...

To overcome this, the user can decide to introduce so-called „key frames", which do not depend on any preceding frames. Both the concept and the term are taken from video encoding, where the same problem exists. As a compromise, such a key frame could be stored every 100 frames. Then, in the worst, case, you would have to decompress 100 frames until you obtain your desired frame. A key frame could be also stored every 10 frames, but this would significantly reduce the compression efficiency *(key frames are larger because they can't exploit knowledge on frame history)*. A key frame interval of 100 is specified by adding „`-keyframe 100`" to the command line. The default is to disable writing of key frames, which is a reasonable setting if only sequential access will be required. To allow for random access, you should change this value.

5. The bqb File Format

5.1. Introduction

Together with our compression algorithm, we also developed a file format to store the compressed data, which we call the bqb format. The name bqb originally stood for „binary cube“ *(referring to Gaussian Cube files)*, but the current bqb format can store also other types of data. Our format is very size-efficient. Even all the headers and control structures are compressed, and the format is built as a stream of bits, not considering byte boundaries. This ensures that not a single bit of storage space is wasted. On the other hand, the format contains „robustness features“ such as magic numbers and CRC-32 codes, such that any corruption of a file will be immediately recognized.

5.2. Why not HDF5?

We are often asked why we did not use HDF5 to store our compressed data. We indeed considered this. However, HDF5 is based on a different philosophy. HDF5 is designed as a multi-purpose format which can handle as many different applications as possible; its primary design goals is its flexibility. Our bqb format developed here is not as flexible, it is only intended for storage of trajectories from computer simulations. On the other hand, the bqb format aims at the maximum compression efficiency, which it achieves by neglecting flexibility to a certain degree. For example, HDF5 only offers compression of the payload data, while all the headers and control structures of the format still occupy significant storage space. The bqb format is completely compressed, and tries to save every single bit even in the headers and control structures. Our format is not even based on byte boundaries; it is a direct representation of a bit stream, which ensures that not a single bit of storage space is heedlessly wasted. Due to these different design philosophies, we came to the conclusion that none of the existing multi-purpose formats such as HDF5 are suitable for our application.

5.3. Index Frame

bqb files typically possess a so-called „index frame“. This concept is heavily inspired by the „catalog“ in the PDF file format. An index frame is essentially a list of frame lengths, types, and offsets, such that fast random access to individual frames is possible without the need for scanning the whole trajectory. The valid index frame is always located at the very end of the file; all index frames in the middle of the file are simply ignored. This comes with a subtle advantage: If additional frames shall be appended to an existing bqb file, they are simply appended in the literal sense, and the old index frame is not modified. A new index frame is written at the end of the appended frames. Therefore, it is possible to append content without having to modify or overwrite/delete any part of the existing old file.

The existence of an index frame is optional; a bqb file is also valid without an index. However, a missing index leads to many complications. Even simple quantities such as the total frame count inside a bqb file are not known without an index *(and to determine it, one would have to traverse the full trajectory, such as in most other trajectory storage formats)*. Therefore, the bqbtool always writes an index after compressing a file *(and this can't be switched off)*.

In case you ask: Yes, the index frame is of course also compressed with the highest possible compression ratio :-)

5.4. File Extensions

There exist five allowed file extensions for data in bqb format. These are „`.bqb`", „`.btr`", „`.blist`", and „`.emp`". These extensions indicate different types of data in the compressed files. However, this is only to guide the user. Technically, every kind of bqb file may carry each of the mentioned extensions. In the following, a short description of each file extension is given, including our recommendations on usage.

`.bqb` is the general file extension, without making a claim about the content of the file. Originally, it was only intended for volumetric data trajectories.

`.btr` stands for „binary trajectory", and shall be used for compressed position trajectories. However, it is totally fine to use .bqb also for position trajectories.

`.blist` is a so-called list file. It is not a bqb file, but rather a text file which contains links to multiple „true" .bqb files after a headline. This is an efficient way to sequentially join several bqb files. We recommend that all list files (and only list files) should carry this file extension to avoid confusion. A description of list files follows below in section 5.5.

`.emp` stands for „electromagnetic properties". It is a special type of bqb file, which stores properties such as charge, electric dipole, electric quadrupole, magnetic dipole, etc. for each atom in the system. TRAVIS **[2,3]** writes these files as intermediates in the process of computing vibrational spectra **[6-8]**. As this format has very rigid specifications, you probably should not use this file extensions for your own compressed files to avoid confusion.

5.5. List Files

Often, one wants to sequentially join several bqb files to a long trajectory (*e. g.*, if the simulation was carried out in multiple batches). An easy way to do so would be to join the individual bqb files on filesystem level („`cat file1.bqb file2.bqb > large.bqb`“). This is in principle valid *(because bqb files are simply a concatenation of individual bqb frames)*, but comes with two disadvantages. First, the index frames of the individual bqb files become invalid, such that the resulting large trajectory does not possess a valid index. Secondly, it requires a factor 2 overhead of disk space *(at least temporary, if the small files are deleted aferwards)*.

A much more elegant solution is to use so-called list files. A list file is not a bqb file, but a text file which contains the string „`BLIST`“ in its first line (all uppercase and without any leading characters). In all following lines, „true“ bqb files can be specified. This is done either by giving a relative path (which is always evaluated relative to the location of the list file) or by giving an absolute path. Nesting of list files is currently not allowed (*i. e.*, a line in a list file may not point to another list file). List files should always carry the file extension „`.blist`“ *(see section 5.4)*.

List files are handled *transparently* in the background. This means that an application which supports the bqb format cannot tell the difference between a list file and a „true“ bqb file; it would behave identically in both cases. If every entry of the list file possesses a valid index frame, it is easy to reconstruct a joint index for the list file from this information. Therefore, the list file behaves as it would possess a valid index in this case, allowing for random access.

Let’s consider a simple example. The following list file invokes three bqb files. The first two are specified by relative path. The first one resides in a subdirectory with respect to the list file. The second one is found in the parent directory of the list file. The third one is named by absolute path.

```
BLIST
subdir/file1.bqb
../file2.bqb
/home/brehm/calc/subdir/file3.bqb
```

6. Contact

If you would like to report a bug or have comments or suggestions on the bqbtool, please feel free to contact the developers via mail:

Martin_Brehm@gmx.de

The mail address is embedded as an image to keep away spam bots. A clickable version (which is of course also well protected (-:) can be found on

`https://brehm-research.de/contact`

7. Literature

[1] M. Brehm, M. Thomas: "An Efficient Lossless Compression Algorithm for Trajectories of Atom Positions and Volumetric Data", *J. Chem. Inf. Model.* **2018**, *in press*, DOI 10.1021/acs.jcim.8b00501

[2] `http://www.travis-analyzer.de`

[3] M. Brehm, B. Kirchner: "TRAVIS - A free Analyzer and Visualizer for Monte Carlo and Molecular Dynamics Trajectories", *J. Chem. Inf. Model.* **2011**, *51 (8)*, pp 2007-2023.

[4] XMol XYZ File Format: http://www.ccl.net/chemistry/resources/messages/1996/10/21.005-dir/index.html

[5] Gaussian Cube File Format: http://paulbourke.net/dataformats/cube/

[6] M. Thomas, M. Brehm, B. Kirchner: "Voronoi dipole moments for the simulation of bulk phase vibrational spectra", *Phys. Chem. Chem. Phys.* **2015**, *17*, pp 3207-3213.

[7] M. Thomas, B. Kirchner: "Classical Magnetic Dipole Moments for the Simulation of Vibrational Circular Dichroism by ab Initio Molecular Dynamics", *J. Phys. Chem. Lett.* **2016**, *7*, pp 509-513.

[8] M. Brehm, M. Thomas: "Computing Bulk Phase Raman Optical Activity Spectra from ab initio Molecular Dynamics Simulations", *J. Phys. Chem. Lett.* **2017**, *8 (14)*, pp 3409-3414.