Genomedata Documentation

Release 1.4.0

Michael M. Hoffman

Aug 16, 2017

Contents

Contents:

CHAPTER 1

Genomedata 1.4 documentation

Website http://pmgenomics.ca/hoffmanlab/proj/genomedata/

Author Michael M. Hoffman <michael dot hoffman at utoronto dot ca>

Organization Princess Margaret Cancer Centre

Address Toronto Medical Discovery Tower 11-311, 101 College St, M5G 1L7, Toronto, Ontario, Canada

For a broad overview, see the paper:

> Hoffman MM, Buske OJ, Noble WS. (2010). The Genomedata format for storing large-scale functional genomics data. *Bioinformatics*, **26**(11):1458-1459; doi:10.1093/bioinformatics/btq164

Please cite this paper if you use Genomedata.

Installation

Python (2.6 or 2.7) and the HDF5 libraries are required before you can install Genomedata.

Installing HDF5

Ubuntu/Debian:

```
sudo apt-get install libhdf5-serial-dev hdf5-tools
```

CentOS/RHEL/Fedora:

```
sudo yum -y install hdf5 hdf5-devel
```

OpenSUSE:

```
sudo zypper in hdf5 hdf5-devel libhdf5
```

If HDF5 has been installed from source, set the HDF5_DIR environment variable to the directory where it was installed.

Installing Numpy

With Python 2.6 or 2.7 installed:

```
pip install numpy
```

Installing Genomedata

With Python 2.6 or 2.7 installed:

```
pip install genomedata
```

Note: The latest version of genomedata may not will not install with older versions of pip (< 6.0) due to some of the dependencies requiring a newer version. You can update your pip using the command:

```
pip install --upgrade pip
```

Note: Genomedata is only supported on 64 bit systems.

Note: The following are prerequisites:

- **Linux/Unix** This software has been tested on Linux and Mac OS X systems. We would love to add support for other systems in the future and will gladly accept any contributions toward this end.
- Zlib

Note: For questions, comments, or troubleshooting, please refer to the *support* section.

Overview

Genomedata provides a way to store and access large-scale functional genomics data in a format which is both space-efficient and allows efficient random-access. Genomedata archives are currently write-once, although we are working to fix this.

Under the surface, Genomedata is *implemented* as one or more HDF5 files, but Genomedata provides a transparent interface to interact with your underlying data without having to worry about the mess of repeatedly parsing large data files or having to keep them in memory for random access.

The Genomedata hierarchy:

Each `Genome` contains many `Chromosomes`

Each `Chromosome` contains many `Supercontigs`

Each `Supercontig` contains one `continuous` data set Each `continuous` data set is a numpy.array of floating point numbers with a column for each data track and a row for each base in the data set.

Why have `Supercontigs`? Genomic data seldom covers the entire genome but instead tends to be defined in large but scattered regions. In order to avoid storing the undefined data between the regions, chromosomes are divided into separate supercontigs when regions of defined data are far enough apart. They also serve as a convenient chunk since they can usually fit entirely in memory.

Implementation

Genomedata archives are implemented as one or more HDF5 files. The *API* handles both single-file and directory archives transparently, but the implementation options exist for several performance reasons.

Use a directory with few chromosomes/scaffolds:

- Parallel load/access
- Smaller file sizes

Use a single file with many chromosomes/scaffolds:

- More efficient access with many chromosomes/scaffolds
- Easier archive distribution

Implementing the archive as a directory makes it easier to parallelize access to the data. In particular, it makes it easy to create the archives in parallel with one chromosome on each machine. It also reduces the likelihood of running into the 2 GB file limit applicable to older applications and older versions of 32-bit UNIX. We are currently using an 81-track Genomedata archive for our research which has a total size of 18 GB, but the largest single file (chr1) is only 1.6 GB.

A directory-based Genomedata archive is not ideal for all circumstances, however, such as when working with genomes with many chromosomes, contigs, or scaffolds. In these situations, a single file implementation would be much more efficient. Additionally, having the archive as a single file allows the archive to be distributed much more easily (without tar/zip/etc).

Note: The default behavior is to implement the Genomedata archive as a directory if there are fewer than 100 sequences being loaded and as a single file otherwise.

New in version 1.1: Single-file-based Genomedata archives

Creation

A Genomedata archive contains sequence and may also contain numerical data associated with that sequence. You can easily load sequence and numerical data into a Genomedata archive with the *genomedata-load* command (see command details additional details):

```
genomedata-load [-t trackname=signalfile]... [-s sequencefile]... GENOMEDATAFILE
```

This command is a user-friendly shortcut to the typical workflow. The underlying commands are still installed and may be used if more fine-grained control is required (for instance, parallel data loading or adding additional tracks later). The commands and required ordering are:

1. *genomedata-load-seq*
2. *genomedata-open-data*
3. *genomedata-load-data*
4. *genomedata-close-data*

Entire data tracks can later be replaced with the following pipeline:

1. *genomedata-erase-data*
2. *genomedata-load-data*
3. *genomedata-close-data*

New in version 1.1: The ability to replace data tracks.

Additional data tracks can be added to an existing archive with the following pipeline:

1. *genomedata-open-data*
2. *genomedata-load-data*
3. *genomedata-close-data*

New in version 1.2: The ability to add data tracks.

As of the current version, Genomedata archives must include the underlying genomic sequence and can only be created with *genomedata-load-seq*. A Genomedata archive can be created without any tracks, however, using the following pipeline:

1. *genomedata-load-seq*
2. *genomedata-close-data*

New in version 1.2: The ability to create an archive without any data tracks.

Note: A call to **`h5repack`** after *genomedata-close-data* may be used to transparently compress the data.

Example

The following is a brief example for creating a Genomedata archive from sequence and signal files.

Given the following two sequence files:

1. A text file, *chr1.fa*:

```
>chr1
taaccctaaccctaaccctaaccctaaccctaaccctaaccctaacccta
accctaaccctaaccctaaccctaaccct
```

2. A compressed text file, *chrY.fa.gz*:

```
>chrY
ctaaccctaaccctaaccctaaccctaaccctaaccctCTGaaagtggac
```

and the following two signal files:

1. *signal_low.wigFix*:

```
fixedStep chrom=chr1 start=5 step=1
0.372
-2.540
0.371
-2.611
0.372
-2.320
```

2. *signal_high.bed.gz*:

```
chrY    0       12      4.67
chrY    20      23      9.24
chr1    1       3       2.71
chr1    3       6       1.61
chr1    6       24      3.14
```

A Genomedata archive (`genomedata.test`) could then be created with the following command:

```
genomedata-load -s chr1.fa -s chrY.fa.gz -t low=signal_low.wigFix \
    -t high=signal_high.bed.gz genomedata.test
```

or the following pipeline:

```
genomedata-load-seq genomedata.test chr1.fa chrY.fa.gz
genomedata-open-data genomedata.test low high
genomedata-load-data genomedata.test low < signal_low.wigFix
zcat signal_high.bed.gz | genomedata-load-data genomedata.test high
genomedata-close-data genomedata.test
```

Note: chr1.fa and chrY.fa.gz could also be combined into a single sequence file with two sequences.

Note: If using a glob syntax for your sequence files, remember to put the glob filename in quotes to avoid having your shell expand the glob before it genomedata-load uses it (e.g. -s "chr*.agp.gz")

Warning: It is important that the sequence names (*chrY*, *chr1*) in the signal files match the sequence identifiers in the sequence files exactly.

Genomedata usage

Python interface

The data in Genomedata is accessed through the hierarchy described in *Overview*. A full *Python API* is also available.

Note: The Python API expects that a genomedata archive has already been created. This can be done manually via the *genomedata-load* command. Alternatively, this can be done programmatically using :_load_seq:*load_seq*.

To appreciate the full benefit of Genomedata, it is most easily used as a contextmanager:

```
from genomedata import Genome
[...]
gdfilename = "/path/to/genomedata/archive"
with Genome(gdfilename) as genome:
    [...]
```

Note: If Genome is used as a context manager, it will clean up any opened Chromosomes automatically. If not, the Genome object (and all opened chromosomes) should be closed manually with a call to `Genome.close()`.

Basic usage

Genomedata is designed to make it easy to get to the data you want.

Here are a few examples:

Get arbitrary sequence (10-bp sequence starting at chr2:1423):

```
>>> chromosome = genome["chr2"]
>>> seq = chromosome.seq[1423:1433]
>>> seq
array([116,  99,  99,  99,  99, 103, 103, 103, 103, 103], dtype=uint8)
>>> seq.tostring()
'tccccggggg'
```

Get arbitrary data (data from first 3 tracks for region chr8:999-1000):

```
>>> chromosome = genome["chr8"]
>>> chromosome[999:1001, 0:3]  # Note the half-open, zero-based indexing
array([[ NaN,  NaN,  NaN],
       [ 3. ,  5.5,  3.5], dtype=float32)
```

Get data for a specific track (specified data in first 5-bp of chr1):

```
>>> chromosome = genome["chr1"]
>>> data = chromosome[0:5, "sample_track"]
>>> data
array([ 47.,  NaN,  NaN,  NaN,  NaN], dtype=float32)
```

Only specified data:

```
>>> from numpy import isfinite
>>> data[isfinite(data)]
array([ 47.], dtype=float32)
```

Note: Specify a slice for the track to keep the data in column form:

```
>>> col_index = chromosome.index_continuous("sample_track")
>>> data = chromosome[0:5, col_index:col_index+1]
```

Command-line interface

Genomedata archives can be created and loaded from the command line with the *genomedata-load* command.

genomedata-load

This is a convenience script that will do everything necessary to create a Genomedata archive. This script takes as input:

- assembly files in either FASTA (`.fa` or `.fa.gz`) format (where the sequence identifiers are the names of the chromosomes/scaffolds to create), or assembly files in AGP format (when used with `--assembly`). This is **mandatory**, despite having an option interface.
- **trackname, datafile pairs (specified as `trackname=datafile`), where:**
  - trackname is a `string` identifier (e.g. `broad.h3k27me3`)
  - datafile contains signal data for this data track in one of the following formats: WIG, BED3+1, bedGraph, or a gzip'd form of any of the preceding
  - the chromosomes/scaffolds referred to in the datafile MUST be identical to those found in the sequence files
- the name of the Genomedata archive to create

See the *full example* for more details.

Command-line usage information:

```
usage: genomedata-load [-h] [-v] [--verbose] -s SEQUENCE -t NAME=FILE
                       [--assembly | --sizes] [-f | -d]
                       GENOMEDATAFILE

Create Genomedata archive named GENOMEDATAFILE by loading
 specified track data and sequences. If GENOMEDATAFILE
 already exists, it will be overwritten.
 --track and --sequence may be repeated to specify
 multiple trackname=trackfile pairings and sequence files,
 respectively.

 Example: genomedata-load -t high=signal.high.wig -t low=signal.low.bed.gz -s chrX.fa␣
↪-s chrY.fa.gz gdarchive

positional arguments:
  gdarchive             genomedata archive

optional arguments:
  -h, --help            show this help message and exit
  -v, --version         show program's version number and exit

Flags:
  --verbose             Print status updates and diagnostic messages

Input data:
  -s SEQUENCE, --sequence SEQUENCE
                        Add the sequence data in the specified file or files
                        (may use UNIX glob wildcard syntax)
  -t NAME=FILE, --track NAME=FILE
                        Add data from FILE as the track NAME, such as: -t
                        signal=signal.wig
  --assembly            sequence files contain assembly (AGP) files instead of
                        sequence
  --sizes               sequence files contain list of sizes instead of
                        sequence
```

```
Implementation:
  -f, --file-mode       If specified, the Genomedata archive will be
                        implemented as a single file, with a separate h5 group
                        for each Chromosome. This is recommended if there are
                        a large number of Chromosomes. The default behavior is
                        to use a single file if there are at least 100
                        Chromosomes being added.
  -d, --directory-mode  If specified, the Genomedata archive will be
                        implemented as a directory, with a separate file for
                        each Chromosome. This is recommended if there are a
                        small number of Chromosomes. The default behavior is
                        to use a directory if there are fewer than 100
                        Chromosomes being added.
```

Alternately, as described in *Overview*, the underlying Python and C load scripts are also accessible for more finely-grained control. This can be especially useful for parallelizing Genomedata loading over a cluster.

You can use wildcards when specifying sequence files, such as in `genomedata-load-seq -s 'chr*.fa'`. You must be sure to quote the wildcards so that they are not expanded by your shell. For most shells, this means using single quotes (`'chr*.fa'`) instead of double quotes (`"chr*.fa"`).

If you aren't going to use the sequence later on, loading the assembly from an AGP file will be faster and take less memory during loading, and disk space afterward.

genomedata-load-seq

This command adds the provided sequence files to the specified Genomedata, archive creating it if it does not already exist. Sequence files should be in FASTA (`.fa` or `.fa.gz`) format. Gaps of >= 100,000 base pairs in the reference sequence, are used to divide the sequence into supercontigs. The FASTA definition line will be used as the name for the chromosomes/scaffolds created within the Genomedata archive and must be consistent between these sequence files and the data loaded later with *genomedata-load-data*. See *this example* for details.

```
usage: genomedata-load-seq [-h] [-v] [-a] [-s] [-f] [-d] [--verbose]
                           GENOMEDATAFILE seqfiles [seqfiles ...]

Start a Genomedata archive at GENOMEDATAFILE with the provided sequences.
SEQFILEs should be in fasta format, and a separate Chromosome will be created
for each definition line.

positional arguments:
  gdarchive             genomedata archive
  seqfiles              sequences in FASTA format

optional arguments:
  -h, --help            show this help message and exit
  -v, --version         show program's version number and exit
  -a, --assembly        SEQFILE contains assembly (AGP) files instead of
                        sequence
  -s, --sizes           SEQFILE contains list of sizes instead of sequence
  -f, --file-mode       If specified, the Genomedata archive will be
                        implemented as a single file, with a separate h5 group
                        for each Chromosome. This is recommended if there are
                        a large number of Chromosomes. The default behavior is
                        to use a single file if there are at least 100
                        Chromosomes being added.
  -d, --directory-mode  If specified, the Genomedata archive will be
                        implemented as a directory, with a separate file for
```

```
                        each Chromosome. This is recommended if there are a
                        small number of Chromosomes. The default behavior is
                        to use a directory if there are fewer than 100
                        Chromosomes being added.
  --verbose             Print status updates and diagnostic messages
```

genomedata-open-data

This command opens the specified tracks in the Genomedata archive, allowing data for those tracks to be loaded with *genomedata-load-data*.

```
usage: genomedata-open-data [-h] [-v] --trackname TRACKNAME [TRACKNAME ...]
                            [--verbose]
                            gdarchive

Open one or more tracks in the specified Genomedata archive.

positional arguments:
  gdarchive             genomedata archive

optional arguments:
  -h, --help            show this help message and exit
  -v, --version         show program's version number and exit
  --trackname TRACKNAME [TRACKNAME ...]
                        tracknames to open
  --verbose             Print status updates and diagnostic messages
```

genomedata-load-data

This command loads data from stdin into Genomedata under the given trackname. The input data must be in one of these supported datatypes: WIG, BED3+1, bedGraph. The chromosome/scaffold references in these files must match the sequence identifiers in the sequence files loaded with *genomedata-load-seq*. See *this example* for details. A `chunk-size` can be specified to control the size of hdf5 chunks (the smallest data read size, like a page size). Larger values of `chunk-size` can increase the level of compression, but they also increase the minimum amount of data that must be read to access a single value.

BED3+1 format is interpreted the same ways as bedGraph, except that the track definition line is not required.

```
Usage: genomedata-load-data [OPTION...] GENOMEDATAFILE TRACKNAME
Loads data into Genomedata format
Takes track data in on stdin

  -c, --chunk-size=NROWS     Chunk hdf5 data into blocks of NROWS. A higher
                             value increases compression but slows random
                             access. Must always be smaller than the max size
                             for a dataset. [default: 10000]
  -?, --help                 Give this help list
      --usage                Give a short usage message
  -V, --version              Print program version

Mandatory or optional arguments to long options are also mandatory or optional
for any corresponding short options.
```

genomedata-close-data

Closes the specified Genomedata arhive.

```
usage: genomedata-close-data [-h] [-v] [--verbose] gdarchive

Compute summary statistics for data in Genomedata archive and ready for
accessing.

positional arguments:
  gdarchive      genomedata archive

optional arguments:
  -h, --help     show this help message and exit
  -v, --version  show program's version number and exit
  --verbose      Print status updates and diagnostic messages
```

genomedata-erase-data

Erases all data associated with the specified tracks, allowing the data to then be replaced. The pipeline for replacing a data track is:

1. *genomedata-erase-data*
2. *genomedata-load-data*
3. *genomedata-close-data*

```
usage: genomedata-erase-data [-h] [-v] --trackname TRACKNAME [TRACKNAME ...]
                             [--verbose]
                             gdarchive

Erase the specified tracks from the Genomedata archive in such a way that the
track data can be replaced (via genomedata-load-data).

positional arguments:
  gdarchive             genomedata archive

optional arguments:
  -h, --help            show this help message and exit
  -v, --version         show program's version number and exit
  --trackname TRACKNAME [TRACKNAME ...]
                        tracknames to erase
  --verbose             Print status updates and diagnostic messages
```

genomedata-info

This command displays information about a genomedata archive. Running the following command:

```
genomedata-info tracknames_continuous genomedata
```

displays the list of continous tracks. Running:

```
genomedata-info contigs genomedata
```

displays the list of contigs in BED format (0-based, half-open indexing).

This command generates a tab-delimited file containing chromosome name and sizes, suitable for use as a UCSC "chrom sizes" file:

```
genomedata-info sizes genomedata
```

```
usage: genomedata-info [-h] [-v]
                       {tracknames,tracknames_continuous,contigs,sizes}
                       gdarchive

Print information about a genomedata archive.

positional arguments:
  {tracknames,tracknames_continuous,contigs,sizes}
                        available commands
  gdarchive             genomedata archive

optional arguments:
  -h, --help            show this help message and exit
  -v, --version         show program's version number and exit
```

genomedata-query

Prints data from a genomedata archive, for the track TRACKNAME, on CHROM, in the region BEGIN-END (0-based, half-open indexing). Intended as a convenience function only; this is much slower than the Python interface, so it should not be used for large regions.

```
usage: genomedata-query [-h] [-v] gdarchive trackname chrom begin end

print data from genomedata archive in specified trackname and coordinates

positional arguments:
  gdarchive      genomedata archive
  trackname      track name
  chrom          chromosome name
  begin          chromosome start
  end            chromosome end

optional arguments:
  -h, --help     show this help message and exit
  -v, --version  show program's version number and exit
```

Python API

The Genomedata package is designed to be used from a variety of scripting languages, but currently only exports the following Python API.

Tips and tricks

If you find yourself creating many Genomedata archives on the same genome, it might be useful to save a copy of an archive after you load sequence, but before you load any data. Obviously, you can only do this if you use

the fine-grained workflow of *genomedata-load-seq*, *genomedata-open-data*, *genomedata-load-data*, and *genomedata-close-data*.

Technical matters

Chunking and chunk cache overhead

Genomedata uses an HDF5 data store. The data is stored in chunks. The chunk size is 10,000 bp and one data track of 32-bit single-precision floats, which makes the chunk 40 kB. Each chunk is gzip compressed so on disk it will be smaller. To read a single position you have to read its entire chunk off of the disk and then decompress it. There is a tradeoff here between latency and throughput. Larger chunk sizes mean more latency but better throughput and better compression.

The only disk storage overhead is that compression is slightly less efficient than compressing the whole binary data file when you break it into chunks. This is far outweighed by the efficient random access capability. If you have different needs, then it should be possible to change the chunk shape (`genomedata.CONTINUOUS_CHUNK_SHAPE`) or compression method (`genomedata._util.FILTERS_GZIP`).

The memory overhead is dominated by the chunk cache defined by PyTables. On the version of PyTables we use, this is 2 MiB. You can change this by setting tables.parameters.CHUNK_CACHE_SIZE.

Bugs

There is currently an interaction between Genomedata and PyTables that can result in the emission of PerformanceWarnings when a Genomedata file is opened. These can be ignored. We would like to fix these at some point.

Support

To stay informed of **new releases**, subscribe to the moderated `genomedata-announce` mailing list (mail volume very low):

> https://listserv.utoronto.ca/cgi-bin/wa?A0=genomedata-announce-l

For **discussion and questions** about the use of the Genomedata system, there is a `genomedata-users` mailing list:

> https://mailman1.u.washington.edu/mailman/listinfo/genomedata-users

For issues related to the use of Genomedata on **Mac OS X**, please use the above mailing list or contact Jay Hesselberth <jay dot hesselberth at ucdenver dot edu>.

If you want to **report a bug or request a feature**, please do so using our issue tracker:

> https://bitbucket.org/hoffmanlab/genomedata/issues

For other support with Genomedata, or to provide feedback, please write contact the authors directly. We are interested in all comments regarding the package and the ease of use of installation and documentation.

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

Python Module Index

g

Index

G