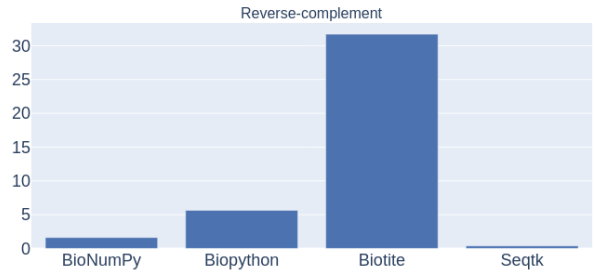
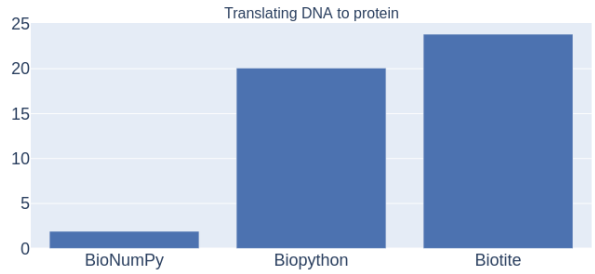
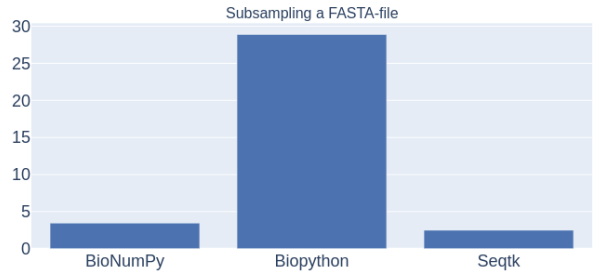
Supplementary material for: Array programming for Biology

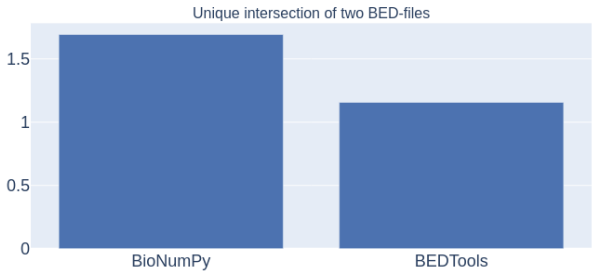
Benchmarks

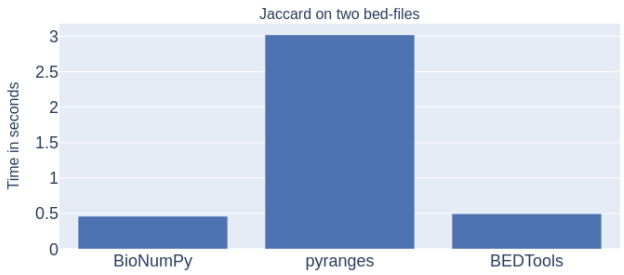
We compare the speed of BioNumPy against other existing Python packages and commonly used non-Python tools on a set of typical bioinformatics tasks. As seen in Figure 1, we find that BioNumPy is generally considerably faster than vanilla Python solutions, as well as the commonly used Python packages BioPython and Biotite, which mostly rely on Python for-loops to perform operations on datasets. On problems where designated efficient bioinformatics tools are widely used (intersection of BED-files, kmer counting and VCF operations), we find that BioNumPy is close to, or as efficient as, tools written in C/C++ (BEDTools [1], Jellyfish [2] and BCFTools [3]). A Snakemake pipeline for reproducing the results can be found at https://github.com/bionumpy/bionumpy/tree/master/benchmarks, along with an open invitation to expand the benchmark with additional tools and cases.

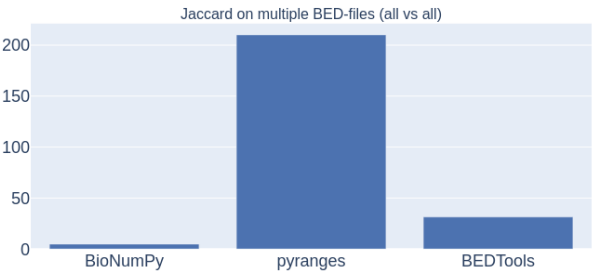

Reverse-complement
30
25
20
15
10
5
0
BioNumPy
Biopython
Biotite
Seqtk

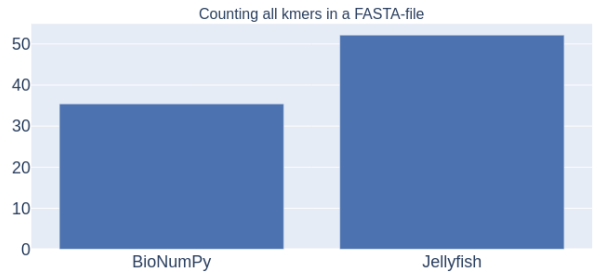

Translating DNA to protein
25
20
15
10
5
0
BioNumPy
Biopython
Biotite

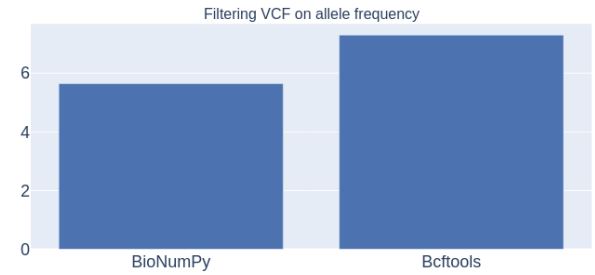

Subsampling a FASTA-file
30
25
20
15
10
5
0
BioNumPy
Biopython
Seqtk

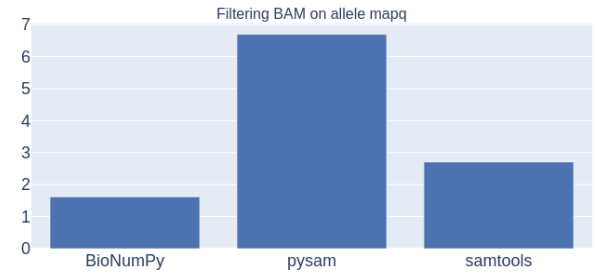

Unique intersection of two BED-files
1.5
1
0.5
0
BioNumPy
BEDTools


Jaccard on two bed-files
Time in seconds
3
2.5
2
1.5
1
0.5
0
BioNumPy
pyranges
BEDTools


Jaccard on multiple BED-files (all vs all)
200
150
100
50
0
BioNumPy
pyranges
BEDTools


Counting all kmers in a FASTA-file
50
40
30
20
10
0
BioNumPy
Jellyfish


Filtering VCF on allele frequency
6
4
2
0
BioNumPy
Bcftools


Filtering BAM on allele mapq
7
6
5
4
3
2
1
0
BioNumPy
pysam
samtools

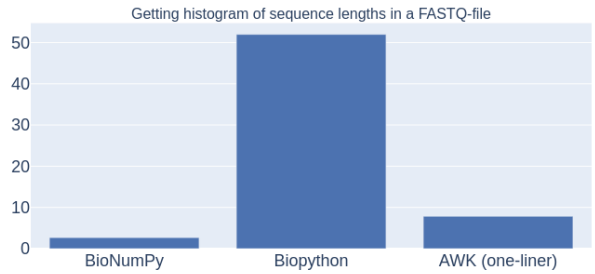

Getting histogram of sequence lengths in a FASTQ-file
50
40
30
20
10
0
BioNumPy
Biopython
AWK (one-liner)

Figure 1: Benchmarking BioNumPy against other tools and methods on various typical bioinformatics tasks.

Reproducing a machine learning benchmark using BioNumPy

TODO

To show how BioNumPy can be used to easily process and parse various biology data formats, we used BioNumPy to reproduce a recently published benchmark of a machine learning method [**sasse?**]. In the original work, the authors are using a combination of custom Python code and common bioinformatics tools (such as BCFTools [3]) to extract sequences around the transcription start sites of genes. We have forked the original repository and replaced all this code, which consisted of X lines of Python code and X lines of shell scripts, with only a few calls to BioNumPy (in total Y lines).

Our fork is available at https://.... We believe this shows that BioNumPy quite easily can be used to cleanly integrate various biology file formats.

BioNumPy Implementation details

BioNumPy internally stores sequence data (e.g. nucleotides or amino acids) as numeric values, allowing the use of standard NumPy arrays for data representation and processing. A key way BioNumpy achieves high performance is by storing multiple data entries in shared NumPy arrays. To illustrate the benefit of this approach, consider the example where we want to count the number of Gs and Cs in a large set of DNA sequences. Existing Python packages like BioPython and Biotite, do this by iterating over the sequences using Python for-loops, which is slow when the number of sequences is large. BioNumPy, however, stores all sequences in only one or a few shared NumPy arrays (Figure 3a), meaning that vectorized NumPy operations can be used to do the counting in a fraction of the time.

Storing multiple elements in shared arrays is trivial if the elements all have the same size, since a matrix representation can be used. However, for biological data, it is common that data elements vary in size. For instance, sequences in FASTA files are rarely all of the exact same size. BioNumPy uses the RaggedArray data structure from the npstructures package (https://github.com/bionumpy/npstructures, developed in tandem with BioNumPy) to tackle this problem (Figure 2). The RaggedArray can be seen as a matrix where rows can have different lengths. The npstructures RaggedArray implementation is compatible with most common NumPy operations, like indexing (Figure 2 b), vectorized operations (Figure 2 c), and reductions (Figure 2 d). As far as possible, objects in BioNumPy follow the array interoperability protocols defined by NumPy (https://numpy.org/doc/stable/user/basics.interoperability.html)

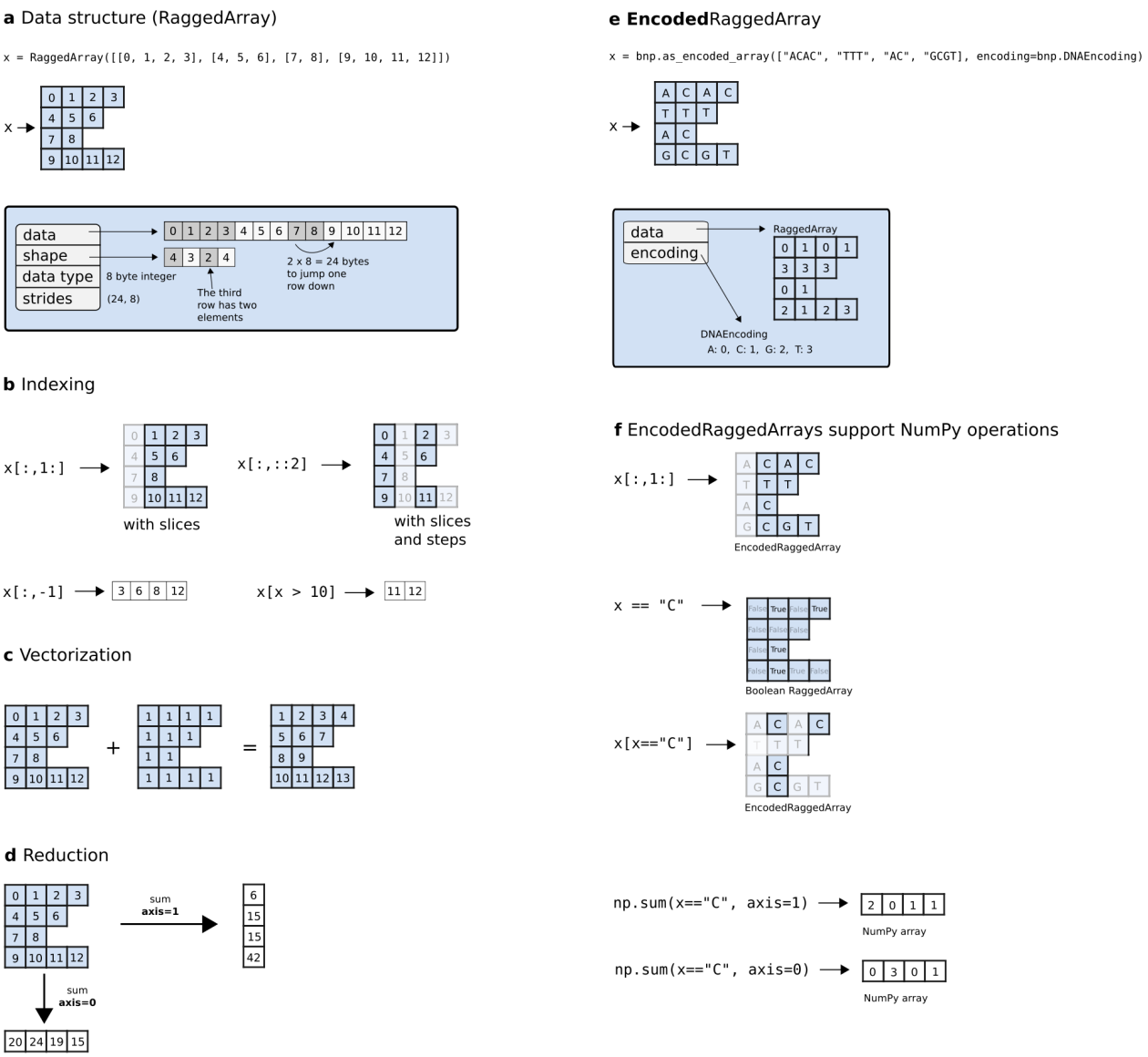


Figure 2: Overview of the RaggedArray and EncodedRaggedArray data structures. A RaggedArray is similar to a NumPy array/matrix but can represent a matrix consisting of rows with varying lengths (a). This makes it able to represent data with varying lengths efficiently in a shared data structure. A RaggedArray supports many of the same operations as NumPy arrays, such as indexing (b), vectorization (c) and reduction (d). An EncodedRaggedArray is a RaggedArray that supports storing and operating on non-numeric data (e.g. DNA sequences) by encoding the data and keeping track of the encoding (e). An EncodedRaggedArray supports the same operations as RaggedArrays (f). This figure is an adopted and modified version of Figure 1 in :cite: `numpy` and is licensed under a Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/).

BioNumPy has been developed following the principles of continuous integration and distribution. The codebase is thoroughly and automatically tested through an extensive collection of unit tests, application tests, integration tests and property-based tests [4]. New code changes are automatically benchmarked and tested before being automatically published, ensuring that updates can be frequent while high code quality is maintained. This workflow makes it safe and easy to allow contributions from new contributors, which is important for longevity and community adoption of the package.

References

1. **BEDTools: a flexible suite of utilities for comparing genomic features**
   Aaron R Quinlan, Ira M Hall
   Bioinformatics (2010-01-28) https://doi.org/cmrms3
   DOI: 10.1093/bioinformatics/btq033 · PMID: 20110278 · PMCID: PMC2832824

2. **A fast, lock-free approach for efficient parallel counting of occurrences of k-mers**
   Guillaume Marçais, Carl Kingsford
   Bioinformatics (2011-01-07) https://doi.org/b7gkd6
   DOI: 10.1093/bioinformatics/btr011 · PMID: 21217122 · PMCID: PMC3051319

3. **Twelve years of SAMtools and BCFtools**
   Petr Danecek, James K Bonfield, Jennifer Liddle, John Marshall, Valeriu Ohan, Martin O Pollard, Andrew Whitwham, Thomas Keane, Shane A McCarthy, Robert M Davies, Heng Li
   GigaScience (2021-01-29) https://doi.org/gjxzc9
   DOI: 10.1093/gigascience/giab008 · PMID: 33590861 · PMCID: PMC7931819

4. **Hypothesis: A new approach to property-based testing**
   David MacIver, Zac Hatfield-Dodds, Many Contributors
   Journal of Open Source Software (2019-11-21) https://doi.org/gs757j
   DOI: 10.21105/joss.01891