BMC Bioinformatics

SOFTWARE **Open Access**

ERStruct: a fast Python package for inferring the number of top principal components from whole genome sequencing data

Jinghan Yang[1†], Yuyang Xu[1†], Minhao Yao[1], Gao Wang[2] and Zhonghua Liu[3*]

†Jinghan Yang and Yuyang Xu contributed equally.

*Correspondence: zl2509@cumc.columbia.edu

[1] Department of Statistics and Actuarial Science, The University of Hong Kong, Pokfulam, Hong Kong SAR, China
[2] Department of Neurology, Gertrude. H. Sergievsky Center, Columbia University, New York, NY, USA
[3] Department of Biostatistics, Columbia University, New York, NY, USA

Abstract

Background: Large-scale multi-ethnic DNA sequencing data is increasingly available owing to decreasing cost of modern sequencing technologies. Inference of the population structure with such sequencing data is fundamentally important. However, the ultra-dimensionality and complicated linkage disequilibrium patterns across the whole genome make it challenging to infer population structure using traditional principal component analysis based methods and software.

Results: We present the ERStruct Python Package, which enables the inference of population structure using whole-genome sequencing data. By leveraging parallel computing and GPU acceleration, our package achieves significant improvements in the speed of matrix operations for large-scale data. Additionally, our package features adaptive data splitting capabilities to facilitate computation on GPUs with limited memory.

Conclusion: Our Python package ERStruct is an efficient and user-friendly tool for estimating the number of top informative principal components that capture population structure from whole genome sequencing data.

Keywords: Population structure, Principal component, Random matrix theory, Sequencing data, Spectral analysis

Background

With the fast development and decreasing cost of next generation sequencing technology, whole genome sequencing (WGS) data are increasingly available and hold the promise of discovering the genetic architecture of human traits and diseases. One fundamental question is to infer population structure from the WGS data, which is critically important in population genetics and genetic association studies [1–3].

PCA (principal component analysis) based methods are prevalent in capturing the population structure from array-based genotype data [4–6]. However, it has been challenging to determine the number of top PCs (principal components) that can sufficiently capture the population structure in practice. A popular traditional method [5] does not perform well on sequencing data for two reasons: ultra-dimensionality [7] and linkage disequilibrium [5]. To resolve those two practical issues on sequencing data, a novel method ERStruct based on eigenvalue ratios has been proposed [8] for whole genome sequencing data and a MATLAB toolbox has been developed. Substantial improvements in accuracy and robustness are found when applying the ERStruct toolbox both on the HapMap 3 project [9] array-based data and the 1000 Genomes Project [10] sequencing data. Despite the great potential, we found that two issues may restrict the use of the ERStruct algorithm:

1 Although the ERStruct MATLAB toolbox provides a parallelization computing feature, its scalability is heavily restricted by the size of memory available in the working environment, slowing down the speed of data analysis.
2 The ERStruct algorithm was implemented as a MATLAB toolbox which is not freely available. In contrast, Python is open-sourced and free to use.

To improve the efficiency and accessibility of the ERStruct algorithm, we develop a new Python package implementing the same ERStruct algorithm. Our ERStruct Python implementation uses parallelization computing to accelerate simulations of GOE (Gaussian Orthogonal Ensemble) matrices used in the ERStruct algorithm. In addition, the package provides optional GPU acceleration to boost the speed of large-scale data matrix operations while maintaining feasible memory usage for GPUs with limited Video Random Access Memory (VRAM). We applied the ERStruct Python package to the 1000 Genomes Project data to demonstrate the computationally efficient performance in the "Results" section. Compared with the original MATLAB version, we achieved a similar time spent using our Python implementation using only CPU (Central Processing Unit) acceleration and significantly reduced time consumption and memory usage with GPU acceleration.

Algorithm

For self-sufficiency of the article, in this section we briefly describe the ERStruct algorithm from [8]. The algorithm starts from a n-by-p genotype data matrix $\mathbf{C}$ that consists of p genetic markers from n individuals. Each of the entry $\mathbf{C}(i,j)$ takes a value from $\{0,1,2\}$, which represents the raw count of the minor alleles for the genetic marker j on the individual i. The ERStruct algorithm estimates the number of top informative PCs that capture the latent population structure. Suppose there are K different (latent) subpopulations in total, and the ith individual is the lth person in the kth subpopulation, then the ERStruct model for the genetic markers of this ith individual is given by

$$\mathbf{C}(i,\cdot) \equiv \mathbf{c}_{k,l} = \mu_k + \varepsilon_{k,l}. \tag{1}$$

In this model, the p-dimensional vectors μ_k denote the kth subpopulation mean counts of minor alleles, and the vectors $\varepsilon_{k,l}$ denote the individual noise vectors, which are independent and identically distributed with mean zeros and an arbitrary covariance matrix Σ.

We now summarize the key steps of the ERStruct algorithm. First, the data matrix $\mathbf{C}$ is normalized column by column into the new matrix $\mathbf{M}$ by

$$\mathbf{M}(i,\cdot) = \left(\hat{\mathbf{D}}\cdot(\mathbf{C}(i,\cdot) - \hat{\mu})\right)^{\top}, \tag{2}$$

where

$$\hat{\mu} = \frac{1}{n}\sum_{i=1}^{n} \mathbf{C}(i,\cdot) = (\hat{\mu}_1, \ldots, \hat{\mu}_p)^{\mathsf{T}}, \qquad \hat{\mathbf{D}} = \operatorname{diag}\left(1\Big/\sqrt{\wp_{\mathrm{j}}(1-\wp_{\mathrm{j}}/2)}\right). \tag{3}$$

Next, we calculate the matrix $\mathbf{S}_p = \mathbf{M}\mathbf{M}^{\mathsf{T}}/p$, compute its non-zero ordered sample eigenvalues $\ell_1 \geqslant \cdots \geqslant \ell_{n-1} > 0$, and obtain the corresponding sample eigenvalue ratios $r_i = \ell_{i+1}/\ell_i$, $i = 1, \ldots, n-2$.

According to the finite-rank perturbation theory [11, 12], under the ERStruct model 1 and the following two assumptions:

(1) Ultra-high-dimensional asymptotic regime: $n \to \infty$ and $n/p \to 0$;
(2) $\min_{k \neq k^*} \|\mu_k - \mu_{k^*}\| \to \infty$;

the sample eigenvalue ratios $r_1, \ldots, r_{n-2}$ can be separated into two sets: the bulk and the spike. The bulk set contains the major part of eigenvalue ratios $r_K, \ldots, r_{n-2}$, which will form a compact set and go to 1 asymptotically. The spike set contains the remaining top $K-1$ sample eigenvalue ratios $r_1, \ldots, r_{K-1}$, which will converge (as $n \to \infty$) to certain limits that are less than 1 and well-separated from the bulk set (see Fig. 1 for illustration).

The ERStruct algorithm estimates the number of top informative PCs K by estimating the number of spikes as follows,

$$\hat{K}_{ER} := \min\left\{1 \leqslant k \leqslant \hat{K}_c \text{ such that } r_k \geqslant \xi_{\alpha,k}, \ldots, r_{\hat{K}_c} \geqslant \xi_{\alpha,\hat{K}_c}\right\}. \tag{4}$$

That is, the first index k such that the kth and the $\hat{K}_c$ subsequent eigenvalue ratios $r_k, \ldots, r_{\hat{K}_c}$ are respectively greater than their critical values $\xi_{\alpha,k}, \ldots, \xi_{\alpha,\hat{K}_c}$, which are the lower α quantiles of the distributions of $r_k, \ldots, r_{\hat{K}_c}$. Note that there are two user-specified parameters in Eq. 4: the significance level α and the coarse estimator $\hat{K}_c$. Based on

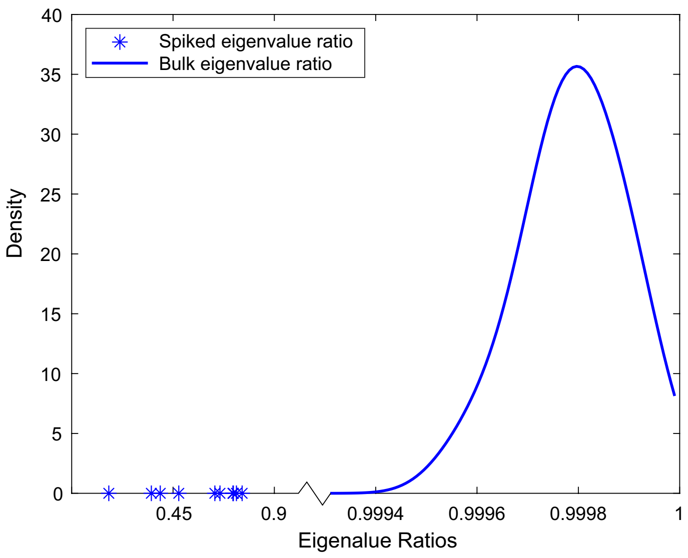


Fig. 1 Illustration of a typical distribution of the eigenvalue ratios under the ERStruct model 1 ($K = 12$, $n = 2500$, $p = 8{,}000{,}000$)

the real data analysis results in [8], we recommend to use $\alpha = 0.001$, and set the coarse estimate $\hat{K}_c = \lfloor n/10 \rfloor$ by default. Note that the coarse estimate should be generally larger than the true number of top PCs K, so as to avoid under-estimation of the original eigenvalue ratio based estimator proposed in [13].

To determine the critical value $\xi_{\alpha,k}$ for r_k, the null distribution of r_k is needed and can be approximated by the distribution of the bulk eigenvalue ratio r_K when the sample size is $n - k + 1$ according to [12] (denotes as $r_K^{(k)}$). Applying the random matrix theory introduced in [14–16], the distribution of $r_K^{(k)}$ can be further approximated by

$$r_K^{(k)} \dot{\sim} \frac{w_2 \cdot \sqrt{\hat{b}_p^{(k)}/p} + \hat{a}_p^{(k)}}{w_1 \cdot \sqrt{\hat{b}_p^{(k)}/p} + \hat{a}_p^{(k)}}, \tag{5}$$

where the symbol $\dot{\sim}$ denotes the left and right sides asymptotically follow the same distribution, and

$$\hat{a}_p^{(k)} = \frac{1}{n-k} \sum_{i=k}^{n-1} \ell_i, \qquad \hat{b}_p^{(k)} = \frac{p}{(n-k)^2} \sum_{i=k}^{n-1} (\ell_i - \hat{a}_p^{(k)})^2, \tag{6}$$

w_1, w_2 are the top two eigenvalues of a n-by-n GOE matrix (i.e., a square matrix with independent entries, where each diagonal entry follows $N(0, 2)$ and each off-diagonal entry follows $N(0, 1)$).

Equations 5 and 6 link the distribution of (w_1, w_2) together to our target distribution of $r_K^{(k)}$, and thus the distribution of r_k. In the ERStruct algorithm, Monte Carlo simulation is used to find out the empirical distribution of (w_1, w_2). Denote all the simulated replications as $(w_1^{(m)}, w_2^{(m)})$, $m = 1, \ldots, rep$, where *rep* is the number of replications. Then the empirical approximated distribution of r_k can be calculated by substituting (w_1, w_2) as $(w_1^{(m)}, w_2^{(m)})$ in Eq. 5, and sorting the results in ascending order (denotes as $r_k^{(1)} \leqslant r_k^{(2)} \leqslant \cdots \leqslant r_k^{(rep)}$). Finally, the critical value can be calculated by

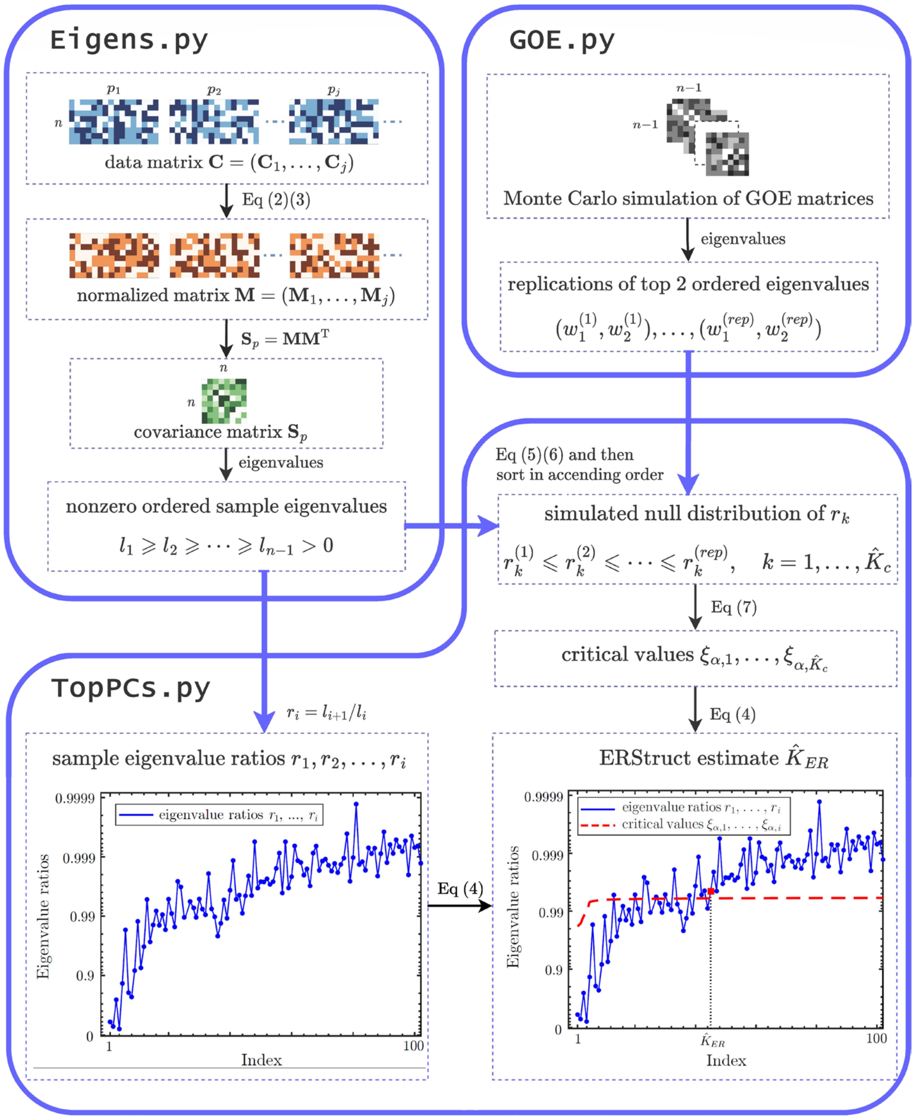
$$\xi_{\alpha,k} = r_k^{(\alpha \cdot rep)}, \tag{7}$$

and the ERStruct estimator $\hat{K}_{ER}$ can be obtained by Eq. 4.

Note that to ensure validity, the number of replications *rep* should be in general greater than $1/\alpha$, the reciprocal of the input significance level. We recommend users to choose between $2/\alpha$ to $5/\alpha$ for the argument *rep*.

Implementation

For a given genotype data matrix as input, the ERStruct Python package estimates the number of top informative PCs that capture the latent population structure in three parts: GOE matrices simulation, calculating the eigenvalue ratios of the given data matrix, and estimating the number of top PCs. These parts correspond to `GOE.py`, `Eigens.py` and `TopPCs.py` files shown in Fig. 2, respectively. To boost the overall performance of the ERStruct algorithm, instead of the frequently used NumPy array processing framework, we choose to build up our ERStruct algorithm using PyTorch tensor and PyTorch functions. According to our experiments, PyTorch functions (e.g., `torch.nanmean`, `torch.nansum`, and `torch.linalg.eigvalsh`) are much



Eigens.py
data matrix $\mathbf{C} = (\mathbf{C}_1, \ldots, \mathbf{C}_j)$
Eq (2)(3)
normalized matrix $\mathbf{M} = (\mathbf{M}_1, \ldots, \mathbf{M}_j)$
$\mathbf{S}_p = \mathbf{M}\mathbf{M}^T$
covariance matrix $\mathbf{S}_p$
eigenvalues
nonzero ordered sample eigenvalues
$l_1 \geqslant l_2 \geqslant \cdots \geqslant l_{n-1} > 0$
GOE.py
Monte Carlo simulation of GOE matrices
eigenvalues
replications of top 2 ordered eigenvalues
$(w_1^{(1)}, w_2^{(1)}), \ldots, (w_1^{(rep)}, w_2^{(rep)})$
Eq (5)(6) and then sort in accending order
simulated null distribution of r_k
$r_k^{(1)} \leqslant r_k^{(2)} \leqslant \cdots \leqslant r_k^{(rep)}, \quad k = 1, \ldots, \hat{K}_c$
Eq (7)
critical values $\xi_{\alpha,1}, \ldots, \xi_{\alpha,\hat{K}_c}$
Eq (4)
ERStruct estimate $\hat{K}_{ER}$
TopPCs.py
$r_i = l_{i+1}/l_i$
sample eigenvalue ratios $r_1, r_2, \ldots, r_i$
Eigenvalue ratios
Index
Eq (4)
eigenvalue ratios $r_1, \ldots, r_i$
critical values $\xi_{\alpha,1}, \ldots, \xi_{\alpha,i}$
$\hat{K}_{ER}$

Fig. 2 A flowchart that demonstrates the three parts of the ERStruct Python package for the whole genome sequencing data analysis. As an example we use the 1000 Genomes Project sequencing data set [10] in which genetic markers with MAF less than 5% are removed. The input real data are processed as in `Eigens.py` and then transmitted to `TopPCs.py` to obtain the sample eigenvalue ratios r_i (as plotted in the lower left panel). While in `GOE.py`, GOE matrices simulation is carried out and then transmitted to `TopPCs.py` to calculate the critical values $\xi_{\alpha,1}, \ldots, \xi_{\alpha,\hat{K}_c}$. Finally, these critical values are used to infer the number of top principal components following Eq. 4 (as plotted in the lower right panel)

faster than their NumPy equivalences (i.e., `numpy.nanmean`, `numpy.nansum`, and `numpy.linalg.eigvalsh`) for data processing in the ERStruct algorithm.

Scalable simulation of GOE matrices

In `GOE.py`, Monte Carlo method is used in the ERStruct algorithm to obtain the null distribution of our proposed ERStruct test statistic, which starts by generating multiple replications of high-dimensional GOE matrices. A significant amount of computing resources is needed in this step, especially when the sample size of the experiment data is large. To efficiently simulate the high-dimensional GOE matrices, we have empirically tested different packages for parallelization computing on different scale GOE matrices, including Joblib, Multiprocessing, and Ray. Joblib is shown to have the most efficient and stable performance on our algorithm to apply parallelization computing for multicore. Without parallelization, the GOE matrices simulation takes 80.68 mins on the function `GOE_L12_sim` with sample size $n = 2504$ and the number of Monte Carlo replications $rep = 5000$, while using parallelization by Joblib on 15 cores CPUs, it takes only 6.46 mins to finish the same job. Compared with the non-parallelization MATLAB version, we successfully decreased the computing time by 12.5 times.

Calculate the eigenvalue ratios of a given matrix

In `Eigens.py`, we start from loading genotype data matrices $\mathbf{C}_i$ from NPY files as the input, where each row of matrix $\mathbf{C}_i$ represents the genetic markers of an individual. Multiple files input are allowed in this step because all of these large-scale data files are often too large to fit in the memory at the same time. In this case, users need to ensure that each data file alone can fit in the memory, and all of these data contain the same individuals in the same order. Also note that by default, our package will impute all the missing data by 0. To achieve a better performance potentially, users may perform other types of imputations beforehand.

The data matrix is then normalized according to Eqs. 2 and 3. In the next a few steps, sample covariance matrix $\mathbf{S}_p = \mathbf{M}\mathbf{M}^\top/p$, non-zero ordered eigenvalues $\ell_1, \dots, \ell_{n-1}$ and eigenvalue ratio $r_i = \ell_{i+1}/\ell_i$ (as shown in the lower left panel in Fig. 2) are calculated accordingly. Although the above matrix operations involve only basic calculations, the data of interest are often massive, occupying a space of hundreds of gigabytes on disk, which may be expensive to process. In our ERStruct Python implementation, we accelerate the above computing steps by converting the CPU Tensor to a CUDA Tensor whenever GPU is available, taking advantage of fast matrix operations in GPU. Limited VRAM capacity on many GPUs makes it difficult to process massive data matrices. However, our approach demonstrates the flexibility of data splitting as a solution to this challenge when working with large data sets on resource-constrained hardware. Our package automatically splits the input data into multiple sub-arrays using a CPU before transmitting it to the GPU. The size of the sub-arrays is determined based on the available VRAM capacity in the working environment. By adopting this approach, users can accelerate computations on large-scale sequencing data, even with a small VRAM GPU, while simultaneously reducing overall memory usage. Our approach highlights the potential of data splitting as an effective method for handling large data sets on hardware with limited resources. Users may use other popular genotype data formats like VCF (variant call format) and bgen files. A tutorial to convert VCF or bgen files to NPY files can be found in the ERStruct code repository.

Estimation of the number of top informative PCs

Finally, the sample eigenvalues $\ell_1, \dots, \ell_{n-1}$ from `Eigens.py` and the top two eigenvalues of simulated GOE matrices $(w_1^{(1)}, w_2^{(1)}), \dots, (w_1^{(rep)}, w_2^{(rep)})$ from `GOE.py` are transmitted together into `TopPCs.py`. Following Eqs. 4–7 in the ERStruct algorithm, the program output the estimation of the number of top informative PCs $\hat{K}_{ER}$ in the end (as shown in the lower right panel in Fig. 2).

Note that if the target estimator $\hat{K}_{ER}$ can not be found through Eq. 4, an error message will appear. This may happen if the data exhibit serious multicollinearity. To resolve this potential issue, users may need to pre-process the genetic data to remove highly correlated genetic markers.

Results

In this section, we compare the speed, maximum memory usage, and accuracy of the original MATLAB [8] and our Python implementations of the ERStruct algorithm. We use the function `tic` in MATLAB and `time.time` in Python to record running time. To record memory usage, we use the Linux shell command `/proc/<pid>/status | grep VmSize` to check memory usage in MATLAB and use the Python module `memory-profiler` for checking in Python. Python (version 3.8.8) is used with NumPy (version 1.20.1), PyTorch (version 1.11.0) and Joblib (version 1.0.1). All results are obtained from a server running x86-64 Linux with 15 Intel(R) Xeon(R) E7-8891 v4 CPU cores and 12 GB Tesla K80 GPU.

We apply our ERStruct Python implementation to the publicly available 1000 Genomes Project [10] data to estimate the number of top informative PCs. The 1000 Genomes Project is a whole genome sequencing data set with 2504 individuals from 26 subpopulations. Following the same procedure in [8], the raw sequencing data file is first filtered out markers with MAF (Minor Allele Frequency) less than 0.05, 0.01, 0.005, and 0.001 using the PLINK software. The remaining number of markers are $p_{0.05}$ = 7,921,8816, $p_{0.01}$ = 13,650,478, $p_{0.005}$ =17,307,567 and $p_{0.001}$ = 28,793,505, respectively. Each preprocessed data is stored as NPY files and tested using our ERStruct Python package and the original MATLAB toolbox by [8]. The testing parameters are fixed as: number of replications $rep = 5000$, significance level $\alpha = 10^{-3}$.

The results are shown in Table 1. The GPU-based Python implementation runs much faster than the CPU-based Python (2.02 times faster when MAF is greater than 0.001) and MATLAB (3.78 times faster when MAF is greater than 0.001) implementations. In terms of the maximum memory usage, the GPU-based Python implementation used only 0.27 of the CPU-based Python implementation and 0.31 of the MATLAB

Table 1 Running time (in minutes) and maximum memory usage (in GB) comparisons of the ERStruct algorithm, using MATLAB, Python CPU, and Python GPU implementations on the 1000 Genomes Project data with different MAF filtering thresholds

MAF filtering thresholds	**0.05**	**0.01**	**0.005**	**0.001**
Running Time				
MATLAB	30.08	42.30	50.05	73.41
Python CPU	34.89	66.54	84.97	137.54
Python GPU	17.60	21.91	28.96	36.42
Maximum Memory Usage				
MATLAB	51.55	62.22	69.54	92.50
Python CPU	28.00	48.80	62.10	104.71
Python GPU	10.08	15.07	18.33	28.67

implementation (when MAF is greater than 0.001) due to the data splitting procedure. Noting that our function `Eigens` automatically splits large-scale data sets so that our algorithm can be run on a GPU with limited VRAM, so using GPU acceleration can be slower than using CPU only when the current available VRAM is too small. In our testing environment, this only happens when the available VRAM is less than 0.17 GB. It is only possible when another process in the working environment has taken up a large part of the available VRAM. In this case, we suggest releasing the VRAM first. In the package, we set the default VRAM value to 0.2 GB to ensure better performance.

To evaluate the accuracy of our ERStruct algorithm implemented in both the Python package and MATLAB toolbox, we conducted 30 identical experiments on CPU and GPU, respectively. We used the 1000 Genomes data set with MAF greater than 0.001, and set the number of replications to $rep = 5000$ and the significance level to $\alpha = 10^{-3}$. In all 30 experiments, the output consistently estimated $\hat{K}_{ER} = 25$. These results indicate that the ERStruct algorithm implemented in both versions are identical and produce the same output.

Conclusion

In this paper, we developed a Python package that employs the ERStruct algorithm to determine the optimal number of top informative PCs in WGS data. By leveraging parallel computing on CPUs and GPU acceleration, our package demonstrates exceptional efficiency in performing matrix operations on large-scale sequencing data sets. To enhance the usability of our Python package across various environments, it features adaptive data splitting capabilities for GPU computation with limited VRAM. We conducted experiments that compared the computation speed of our ERStruct Python package to the ERStruct MATLAB toolbox in [8]. Our results demonstrate a significant improvement in computation speed with the ERStruct Python package.

Availability and requirements

Project name: ERStruct.
Project home page: https://github.com/ecielyang/ERStruct.
Programming language: Python.
Other requirements: Python 3.7—3.9, PyTorch 1.10 or higher, NumPy, Joblib.
License: MIT License.

Abbreviations

ERStruct	Eigenvalue-Ratio-based estimator to infer latent population Structure
PCA	Principal Component Analysis
PC	Principal Component
CPU	Central Processing Unit
GPU	Graphics Processing Unit
WGS	Whole Genome Sequencing
GOE	Gaussian Orthogonal Ensemble
VRAM	Video Random Access Memory
MAF	Minor Allele Frequency

Acknowledgments
The authors thank The University of Hong Kong for providing the computing resources.

Author contributions
JY implemented the package, experiments and drafted the manuscript. YX conceived the ideas and drafted the manuscript. ZL conceived the ideas, supervised the manuscript writing and edited the manuscript. MY tested the package on GPU. GW helped with the manuscript revision. All authors read and approved the final manuscript.

Funding
Dr. Gao Wang is supported by grant R01 AG076901.

Availability of data and materials
The data sets generated and/or analyzed during the current study are available in the: https://www.internationalgenome.org/data.

Declarations

Ethics approval and consent to participate
Not applicable.

Consent for publication
Not applicable.

Competing interests
The authors declare that they have no competing interests.

Received: 7 November 2022 Accepted: 25 April 2023
Published online: 02 May 2023

References
1. Price AL, Patterson NJ, Plenge RM, Weinblatt ME, Shadick NA, Reich D. Principal components analysis corrects for stratification in genome-wide association studies. Nat Genet. 2006;38(8):904.
2. Mathieson I, McVean G. Differential confounding of rare and common variants in spatially structured populations. Nat Genet. 2012;44(3):243–6.
3. Wang C, Zhan X, Bragg-Gresham J, Kang HM, Stambolian D, Chew EY, et al. Ancestry estimation and control of population stratification for sequence-based association studies. Nat Genet. 2014;46(4):409–15.
4. Menozzi P, Piazza A, Cavalli-Sforza L. Synthetic maps of human gene frequencies in Europeans. Science. 1978;201:786–92.
5. Patterson N, Price AL, Reich D. Population structure and eigenanalysis. PLoS Genet. 2006;2: e190.
6. Reich D, Price AL, Patterson N. Principal component analysis of genetic data. Nat Genet. 2008;40:491–2.
7. Johnstone IM. On the distribution of the largest eigenvalue in principal components analysis. Ann Stat. 2001;29(2):295–327.
8. Xu Y, Liu Z, Yao J. An eigenvalue ratio approach to inferring population structure from whole genome sequencing data. Biometrics. 2022. https://doi.org/10.1111/biom.13691.
9. The International HapMap 3 Consortium. Integrating common and rare genetic variation in diverse human populations. Nature. 2010;467:52–8.
10. The 1000 Genomes Project Consortium. A global reference for human genetic variation. Nature. 2015;526(7571):68–74.
11. Benaych-Georges F, Nadakuditi RR. The eigenvalues and eigenvectors of finite, low rank perturbations of large random matrices. Adv Math. 2011;227(1):494–521.
12. Benaych-Georges F, Guionnet A, Maida M. Fluctuations of the extreme eigenvalues of finite rank deformations of random matrices. Electron J Probab. 2011;16(60):1621–62.
13. Li Z, Wang Q, Yao J. Identifying the number of factors from singular values of a large sample auto-covariance matrix. Ann Stat. 2017;45(1):257–88.
14. Wigner EP. On the distribution of the roots of certain symmetric matrices. Ann Math. 1958;67(2):325–7.
15. Arnold L. On Wigner's semicircle law for the eigenvalues of random matrices. Probab Theory Relat Fields. 1971;19(3):191–8.
16. Wang L, Paul D. Limiting spectral distribution of renormalized separable sample covariance matrices when $p/n \to 0$. J Multivar Anal. 2014;126:25–52.