ERStruct: A Python Package for Inferring the Number of Top Principal Components from Whole Genome Sequencing Data

Jinghan Yang* Yuyang Xu*
Department of Statistics and Actuarial Science
The University of Hong Kong
Hong Kong SAR, China

Zhonghua Liu†
zl2509@cumc.columbia.edu
Department of Biostatistics
Columbia University
New York, NY, USA

August 15, 2022

Abstract

It is important but challenging to determine the number of principal components (PCs) that can sufficiently capture population structure from whole genome sequencing (WGS) data. We developed the ERStruct Python package to determine the number of top informative PCs from WGS data, which uses parallel GPU computing to boost the speed and is thus scalable.

1 Introduction

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

*The authors wish it to be known that, in their opinion, the first two authors should be regarded as Joint Authors.

†To whom correspondence should be addressed.

PCA-based methods are prevalent in capturing the population structure from array-based genotype data (Menozzi *et al.*, 1978; Patterson *et al.*, 2006; Reich *et al.*, 2008). However, it has been a challenging task to determine the number of top PCs that can sufficiently capture the population structure in practice. Patterson *et al.* (2006) proposed a way to determine and select top PCs. But the method does not perform well on sequencing data for two reasons: ultra-dimensionality and linkage disequilibrium (Xu *et al.*, 2022). To resolve those two practical issues on sequencing data, Xu *et al.* (2022) proposed an algorithm called ERStruct and offered a basic toolbox implementation through MATLAB. They showed a substantial improvement in accuracy and robustness when applying the ERStruct toolbox on the 1000 Genomes Project sequencing data (The 1000 Genomes Project Consortium, 2015). 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 MATLAB version of ERStruct is not free, while python is open-sourced and free to use.

To make the ERStruct algorithm perform more efficiently and become more easily accessible, we develop the ERStruct Python package implementing the same algorithm. Our ERStruct Python implementation uses parallelization computing to accelerate simulations of GOE (Gaussian Orthogonal Ensemble matrix) matrices used in the ERStruct algorithm, and at the same time provides optional GPU acceleration to boost the speed of large-scale data matrix operations while maintaining a small memory usage. We applied the ERStruct Python package to the 1000 Genomes Project data to demonstrate the computationally efficient performance in Section 3. Compared with the original MATLAB version, we achieved a similar time-spend using our Python implementation using only CPU acceleration and significantly reduced time consumption and memory usage with GPU acceleration.

2 Implementation Details

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

The remaining overhead of the ERStruct algorithm occurs mainly in two parts: (1) the simulation of GOE matrices; and (2) the large-scale data matrix operations. In what follows, we will describe the details of how we respectively boost the algorithm performance in the above two parts.

2.1 Scalable Simulation of GOE Matrices

In order to obtain the null distribution of our proposed ERStruct test statistic, Monte Carlo method is used in the ERStruct algorithm, which starts by generating multiple replications of high-dimensional GOE matrices. A great 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 compared 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 matrix simulation takes 80.68 minutes on the function `GOE_L12_sim` with sample size $n = 2504$ and the number of replications $rep = 5000$. With parallelization by Joblib on 15 cores CPUs, it takes only 6.46 minutes. Compared with the non-parallelization version, the parallelization decreases the computing time by 12.5 times.

2.2 GPU acceleration for large-scale data matrix operations

Large-scale data sets usually occupy a space of hundreds of gigabytes on disk, which are prohibitively intensive to process and time-consuming to analyze. In our ERStruct Python implementation, users can accelerate data matrix operations by converting a CPU Tensor to a CUDA Tensor if GPU is available. The major problem here is the limited VRAM (Video Random Access Memory) of different GPUs, which are usually not possible to fit a large-scale data matrix. To resolve it, input data is split into multiple sub-arrays by CPU before it is transmitted to GPU. The split data size depends on how large the VRAM is available in the working environment, which ensures that users can accelerate the computation on large-scale sequencing data even with a small VRAM GPU and simultaneously reduces total memory usage.

3 Results

In this section, we compare the speed and the maximum memory usage of MATLAB and 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 (The 1000 Genomes Project Consortium, 2015) data to estimate the number of top informative PCs. The 1000 Genomes Project is a whole genome sequencing dataset with 2504 individuals from 26 subpopulations. Following the same procedure in Xu *et al.* (2022), the raw sequencing data file is first filtered out markers with MAF (Minor Allele

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		0.05	0.01	0.005	0.001
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
Memory	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

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 pre-processed data is stored as NPY files and tested using our ERStruct Python package and the original MATLAB toolbox by Xu *et al.* (2022). The testing parameters are fixed as: number of replications `rep` = 5000, significance level $\alpha = 10^{-4}$.

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 greater than 0.001) and MATLAB (3.78 times faster when MAF 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 implementation (when MAF 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 freeing the VRAM first.

4 Conclusion

The ERStruct algorithm is a useful method for determining the number of PCs in WGS data. We developed the Python package with NumPy and PyTorch for matrix operations and GPU acceleration, which improves the computational efficiency compared with the MATLAB implementation by Xu *et al.* (2022). To expand the usage of our Python package in different environments, our package adaptively splits large datasets for computation on GPUs with limited VRAM.

Availability

The datasets were derived from sources in the public domain: https://www.internationalgenome.org/data.

Our Python implementation of the ERStruct Algorithm is freely available at https://github.com/ecielyang/ERStruct.

References

Menozzi, P. *et al.* (1978) Synthetic maps of human gene frequencies in Europeans. *Science*, **201**, 786–792.

Patterson, N. *et al.* (2006) Population structure and eigenanalysis. *PLoS Genetics*, **2**, e190.

Reich, D. *et al.* (2008) Principal component analysis of genetic data. *PLoS Genetics*, **2**, 491–492.

The 1000 Genomes Project Consortium (2015) A global reference for human genetic variation. *Nature*, **526**, 68–74.

Xu, Y. *et al.* (2022) An eigenvalue ratio approach to inferring population structure from whole genome sequencing data. *Biometrics*, **64**, 1–23.