PCAone: fast and accurate out-of-core PCA framework for large scale biobank data

Zilong Li[1,*], Jonas Meisner[2,3], and Anders Albrechtsen[1,*]

[1]Section for Computational and RNA Biology, Department of Biology, University of Copenhagen
[2]Biological and Precision Psychiatry, Mental Health Centre Copenhagen, Copenhagen University Hospital
[3]Novo Nordisk Foundation Center for Protein Research, University of Copenhagen
[*]To whom correspondence should be addressed

November 15, 2022

Abstract

Principal component analysis (PCA) is a widely used dimensionality reduction technique in statistics, machine learning and genomics. Fast PCA methods, such as the accurate Implicitly Restarted Arnoldi Method (IRAM) and the fast Randomized Singular Value Decomposition (RSVD), are gaining popularity for capturing the top principal components efficiently. However, a fast and accurate PCA method with low memory usage is needed due to ever growing sample sizes. Standard RSVD algorithms use the whole data matrix in each optimization step which makes out-of-core implementations slow. Here we present PCAone, a RSVD algorithm with window based optimization scheme which allows the algorithm to converge within a few passes through the data. Using large-scale genetic data from the UK Biobank, we show that PCAone is much faster than existing state-of-the-art tools while using a negligible amount of memory and maintaining an accuracy comparable to the much slower IRAM algorithm. Applying PCAone to the imputed genotypes of the UK Biobank (around 0.5 million individuals and 6.1 millions common SNPs) took 9 hours using 18.64 GBs memory and 20 CPU threads to calculate the top 40 PCs accurately which captured population structure, signals of selection (*SLC45A2*), structural variants and low recombination regions. PCAone applied to single-cell RNA sequencing data with 1.3 million mice brain cells took only 47 minutes to capture the top 40 PCs accurately, which is 10x faster than the fastest out-of-core PCA method for scRNA data. PCAone is open source and available at `https://github.com/Zilong-Li/PCAone`.

1 Introduction

Principal component analysis (PCA) is a popular approach and is often used to summarize the relationship between high dimensional observations, such as inferring population structure in genetics (Patterson et al. 2006) and clustering cells from single-cell sequencing experiments (Kiselev et al. 2019). The estimated principal components (PCs) are also widely used in downstream analyses. For example the PCs can be used as covariates in regression to account for confounding factors such as population structure (Price, Patterson, et al. 2006). Popular visualization methods, T-distributed Stochastic Neighbor Embedding (T-SNE) and Uniform Manifold Approximation and Projection (UMAP), are usually applied on the top PCs instead of the full data in order to reduce the number of dimensions to a reasonable amount (Diaz-Papkovich et al. 2019; Maaten et al. 2008). PCA as a general dimensionality reduction technique is also used to process the input data by many existing statistics or machine learning models in both supervised and unsupervised settings. However, full-rank Singular Value Decomposition (SVD) is not computationally feasible when hundreds of thousands of observations are available (Tsuyuzaki et al. 2020; Privé, Luu, et al.

2020). To alleviate the computational burden, fast PCA methods have been proposed to approximate the top PCs efficiently instead of the full-rank SVD and these method are widely used in both genetics and single-cell RNA sequencing (scRNAs) data. These faster methods are making trade-offs between the accuracy, speed and memory usage and are usually only used to approximate a low number of the top PCs. Recently, Tsuyuzaki et al. 2020 systematically reviewed commonly used PCA algorithms. The most widely used fast PCA implementations are either based on the Implicitly Restarted Arnoldi Method (IRAM) (Lehoucq and Sorensen 1996), such as FlashPCA2 (Abraham et al. 2017), bigsnpr (Privé, Aschard, et al. 2018), and OnlinePCA.jl-arnoldi (Tsuyuzaki et al. 2020), or they are based on Randomized Singular Value Decomposition (RSVD) (Halko et al. 2011), such as FastPCA(Galinsky et al. 2016), TeraPCA (Bose et al. 2019) and OnlinePCA.jl-halko (Tsuyuzaki et al. 2020). Other fast methods also exists that can deal with missing data such as PCAngsd (Meisner and Albrechtsen 2018), EMU (Meisner, Liu, et al. 2021) and ProPCA (Agrawal et al. 2020).

IRAM as implemented first in the well known ARPACK package (Lehoucq, Sorensen, and Yang 1998) is accurate but not pass-efficient, which has led to the more recent RSVD gaining popularity despite being less accurate. However, the accuracy of RSVD can be improved by leveraging so called *power iterations* which essentially is an optimization step on the initial random matrix. This, however, introduces more passes over the data which makes it computationally less efficient especially if the data cannot fit in memory. How to efficiently conduct PCA on large-scale and high-dimensional dense data in a limited-memory computation environment is still an open problem. Algorithms that do not require storing all data in memory (out-of-core) can tackle the issue of limited memory, but they are potentially slower due to the I/O operation cost of parsing data stored on disk or stream (Yu et al. 2017; Abraham et al. 2017; Bose et al. 2019). Therefore, a pass-efficient implementation is essential for the out-of-core algorithms such that the amount of times the data is read from the disk is limited.

In this work we present a new algorithm PCAone, a fast and accurate out-of-core framework written in C++ which improves both the accuracy and the speed of the fast RSVD approach as well as including efficient PCA implementations of existing algorithms. The software is designed to process large-scale and high-dimensional data of different formats efficiently for biological researches. Firstly, as RSVD is more pass-efficient but less accurate than IRAM, we proposed and implemented a new RSVD algorithm that can achieve high accuracy within a few passes over the data which makes the implementation well suited for out-of-core computations, where I/O operations, i.e. reading data from the disk, are otherwise the limiting factor. Secondly, we implemented a RSVD algorithm called PCAone$_{H+Y}$ based on Yu et al. 2017 but with a power iteration scheme to improve accuracy. This approach, uses single pass over the data for each power iteration instead of two passes which reduces the amount of I/O compared to other RSVD implementations. In order to compare our algorithm to existing algorithms, we also made efficient implementations of the accurate IRAM algorithm which we refer to as PCAone$_{Arnoldi}$. These methods are implemented in two ways, i.e. in memory and out-of-core, so that the user can achieve higher speed if they have enough memory. However, for this paper we will focus only on the out-of-core implementation, which is the available option for the large scale datasets.

2 Results

2.1 Overview of PCAone framework

We implemented three out-of-core algorithms in our software PCAone. An IRAM algorithm, PCAone$_{Arnoldi}$, inspired by the blockwise approach in FlashPCA2 (Abraham et al. 2017) but with multithreading capabilities. Likewise, we implemented the much faster RSVD algorithm, PCAone$_{H+Y}$, which is based on a single-pass approach proposed by Yu et al. 2017 that reduces the amount of times reading through the data by 50% compared to the original Halko's algorithm.

However, RSVD without power iterations can have low accuracy. Therefore, we implemented a version of the algorithm with power iterations. However, for some data sets many power iterations are needed. Therefore, we also propose a new RSVD algorithm, called PCAone, which used a window based power iteration scheme on permuted subsets of the input matrix. This allows the algorithm to perform multiple power iterations for each pass over the data as illustrated in Figure S2 and Figure S1 (Step 1 PCAone). This is in contrast to the excisting RSVD algorithm where the entire input matrix is read from the disk for each power iteration (Figure S1, Step 1 Halko). The details of PCAone algorithms and architecture is given in Methods and supplementary Figure S3.

2.2 Accuracy and iterations

We compared our three implemented algorithms (PCAone, PCAone$_{H+Y}$, PCAone$_{Arnoldi}$) with software for performing large-scale PCA in genetics, FlashPCA2, FastPCA, Plink2(FastPCA), TeraPCA and ProPCA. Plink2(FastPCA) is a faster reimplimentation of FastPCA (Chang et al. 2015)). To measure the overall accuracy of all estimated PCs, we compute the mean explained variance (MEV) between the estimated PCs and that of the full-rank SVD. We first evaluated the performance on genetic datasets that are small enough for us to compare the different methods with the true PCA from full-rank SVD. The performance of the different methods will depend on the number of samples, number of features e.g. SNPs, as well as the number of top PCs to be estimated and the distribution of eigenvalues.

First, we analyzed a dataset of four genetically similar East Asian populations from the 1000 Genomes Project namely two Han Chinese population (CHB, CHS), one Dai Chinese (CDX) and one Vietnamese (KHV) population each consisting of around 100 individuals and using 2 million common SNPs. Because the four East Asian populations are genetically similar the eigenvalues of the dataset decay slowly (Figure S4), we expect that more iterations are needed to obtain high accuracy (Halko et al. 2011; Yu et al. 2017). The number of times the data is read from the disk(epochs) to estimate the top K PCs are shown in Figure 1 (A). As can be seen, our proposed PCAone algorithm consistently uses 7 epochs for all choices of$K$. The other algorithms use more epochs for higher K, which is determined by both the algorithms convergence efficiency and their stopping criteria. Most methods produce almost identical top 2 PCs for $K = 2$ compared to the full-rank SVD with the exception of Plink2(FastPCA) and ProPCA which failed to accurately capture the difference between the Dai and Vietnamese (Figure 1 B). The main reason for their low accuracy for $K = 2$ is their stopping criteria which is based only on the choice of K and with higher choices of K their accuracy increases. Most of the methods memory usage is small for this data sets and near constant in K with the exception of Plink2(FastPCA) which increase linearly (Figure 1 C). We compared the accuracy across different choices of K (Figure 1 D) which showed that PCAone and the two IRAM methods (PCAone$_{Arnoldi}$, FlashPCA2) consistently had high accuracy while the other methods performed poorly for some choices of K.The difference in accuracy of the RSVD methods can largely be explained by their stopping criteria and thus the number of epoch they use. The notable exception is PCAone which converges faster per epoch than the other the standard single power iteration (Figure 1 E). The faster convergence of PCAone is achieved by performing many power iterations each epochs on subsets of the data.

We also analyzed all populations from 1000 Genomes Project where there is much more pronounced population structure which leads to more of the variation being explained by the top PCs. All methods were more accurate (MEV>0.95) across a range of chosen number of PCs (Figure S5) and PCAone again required only 7 or less epochs to achieve high accuracy.

A

Number of passes over data (epochs) of each method for top K PCs

K(PCs)	PCAone	PCAone$_{H+Y}$	TeraPCA	Plink2(FastPCA)*	ProPCA*	FlashPCA2	PCAone$_{Arnoldi}$
2	7	20	34	2	4	37	38
5	7	39	40	5	7	92	59
10	7	33	44	10	12	89	86
20	7	30	48	20	22	159	140
30	7	33	36	30	32	242	199
40	7	31	36	40	42	253	217

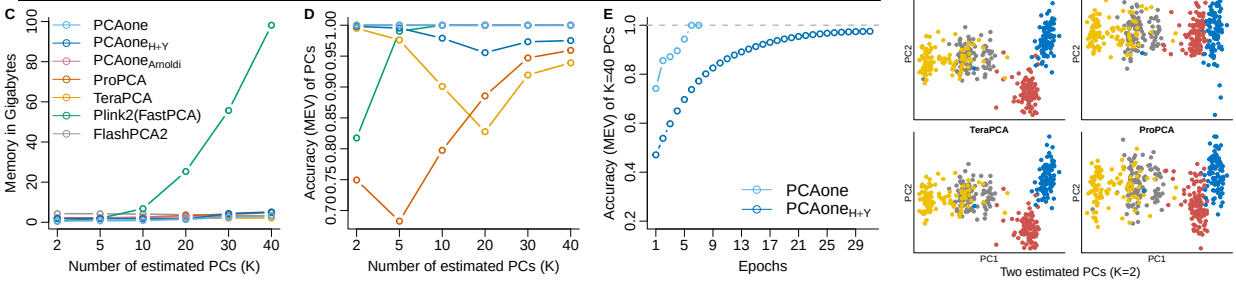

Figure 1: Performance on 1000 Genomes Project East Asian dataset. PCA performance by different software with varying number of inferred PCs (K) based on 2,000,000 SNPs and 400 individuals from 4 closely related east Asian populations. **A**, Number of epochs used by each software. * Not out-of-core, only allows for in memory computation with this number of iterations. **B**, Two estimated PCs $K = 2$ for each methods including the true full-rank SVD. **C**, Memory usage as a function of K. **D**, Accuracy (MEV) compared to the true full-rank SVD as a function of K. **E**, Convergence of PCAone and PCAone$_{H+Y}$ shown as accuracy per epoch.

Finally we tested all methods on subsets of the UK Biobank imputed genetic data where we sampled small enough subsets of individuals and SNPs to run all method. We were only able to run the full-rank SVD on data with 50,000 samples and 3,000,000 SNPs. For larger subsets we used the IRAM method as the baseline to calculate accuracy as it is the most accurate method for capturing top PCs (Privé, Aschard, et al. 2018; Tsuyuzaki et al. 2020). On this data all methods had high accuracy (MEV >0.999) (see Table S1 and low minimum sum of squared error (minSSE)Table S2) which makes it useful for comparison of speed and memory usage.

2.3 Runtime and memory usage

We assessed the scalability and performance of PCAone with various large real-world datasets that are random subsets from the UK Biobank genetic data. We compared the wall-clock time and memory usage to calculate the top $K = 40$ PCs. As shown in Figure 2 , all methods had a mostly linear relationship between speed and number of individuals or SNPs. The memory usage was near constant for the out-of-core implementations (PCAone, PCAone$_{H+Y}$, PCAone$_{Arnoldi}$, FlashPCA2, TeraPCA) and linear for ProPCA and Plink2. Using 20 thread our IRAM implementation, PCAone$_{Arnoldi}$, was > 10 times faster than FlashPCA2. Similarly our out-of-core RSVD implementations, PCAone and PCAone$_{H+Y}$, were faster than the other methods including the in memory methods Plink2(FastPCA) and ProPCA which does not read in data from the disk at each iteration. The fastest method, PCAone, is on average 2x, 5x, 9x, 12x, 30x faster than PCAone$_{H+Y}$, Plink2, ProPCA, TeraPCA, FlashPCA2, respectively, while using only little memory.

We further investigated the performance on the 487,409 individuals and 6,133,304 common SNPs from the UK Biobank. It was not feasible to run ProPCA and Plink2(FastPCA) as they needed more than 180 GBs allocated memory. The accuracy and resource usage are summarized in Table 1. Only PCAone and PCAone$_{H+Y}$ finished within a day with a runtime of 9.2 hours and 12.6 hours, respectively, including time of permuting data required by PCAone, which took around one hour. However, it took 49 hours for TeraPCA and 102 hours for PCAone$_{Arnoldi}$ to finish while we did not expect FlashPCA2 to finish within a month. Also, we calculated MEV and minSSE for the three RSVD implementations compared to PCs estimated by the very accurate IRAM method PCAone$_{Arnoldi}$, which showed PCAone$_{H+Y}$ and TeraPCA had slighty lower MEV and higher minSSE compared to PCAone. We also ran PCAone$_{H+Y}$ with a stricter stopping criteria

to see how many epochs it would take to achieve the same accuracy as PCAone. To show that the slightly lower accuracy had an clearly observable effect on the PCs, we showed the minSSE per PC and highlighted the least accurate individual points for each methods (S7 and S8). As shown by the plots, PCAone is the only method that using default stopping criteria gives visually indistinguishable results to the slower IRAM method.

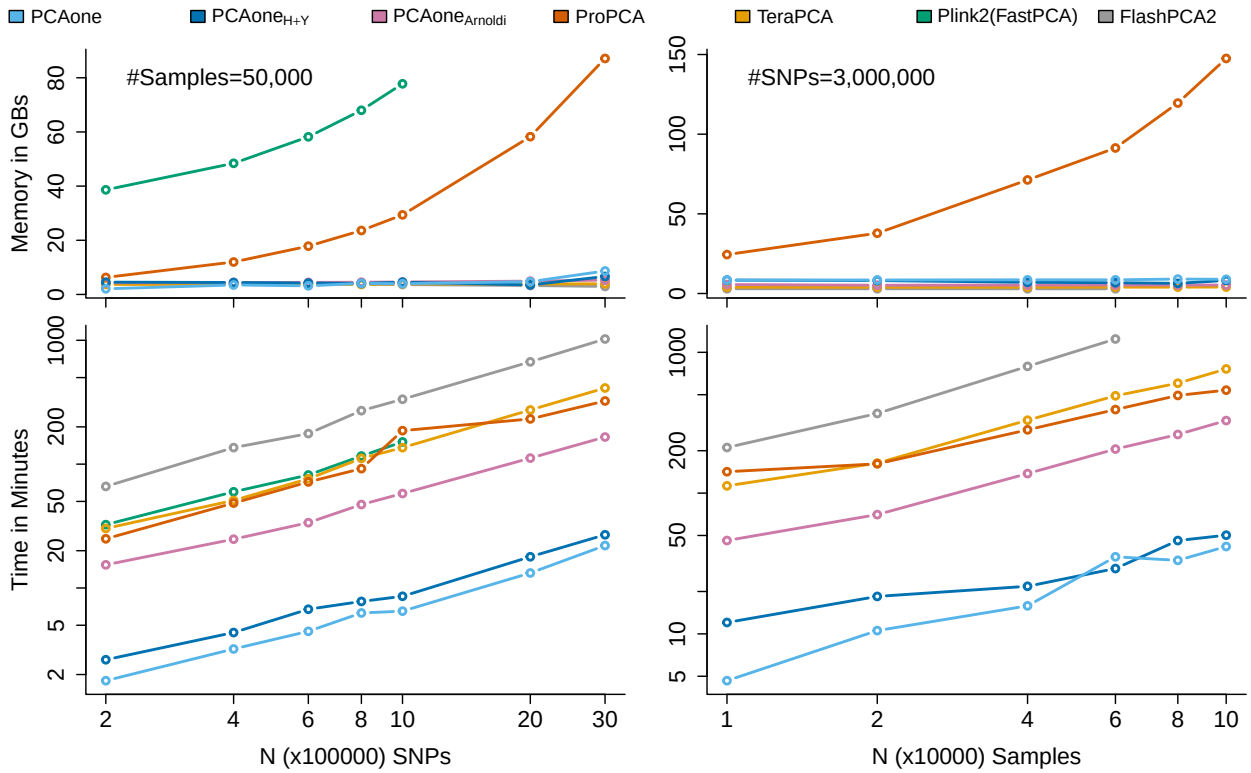

Figure 2: Runtime and memory usage in UK biobank subsets. Runtime and memory usage for calculating top K=40 PCs in random subsets of either common (MAF>5%) SNPs (left column) for 50,000 individuals or random subsets of individuals (right column) for 3,000,000 common SNPs. We use 20 CPU threads for all possible programs but FlashPCA2. Plink2(FastPCA) run out of memory when number of SNPs is greater than 1,000,000.

Table 1: Performance on UK Biobank imputed data with 487,409 individuals and 6,133,304 SNPs for estimating $K = 40$ PCs using 20 CPU threads. Time is given hours and RAM is in gigabytes. α IO time for permuting data. Wall time includes all IO time. PCs estimated by PCAone$_{Arnoldi}$ are used as the true value for calculating accuracy (MEV and minSSE). PCAone$_{H+Y}$* used non default stopping criteria to achieve similar accuracy as PCAone. Accuracy of individual PCs are given in figure S7 and S8.

Program	Wall Time(h)	IO Time(h)	Epochs	RAM(GB)	MEV	minSSE
PCAone	9.2	4.9+0.9α	7	18.64	0.999996	0.000362
PCAone$_{H+Y}$	12.6	6.2	10	17.09	0.999664	0.046124
PCAone$_{H+Y}$*	23.3	16.1	17	17.11	0.999997	0.000223
TeraPCA	49.3	38.8	20	9.40	0.999432	0.048754
PCAone$_{Arnoldi}$	102.7	85.8	124	23.27	-	-

Plink2(FastPCA) and ProPCA couldn't run in limited memory (>180 GBs).

FlashPCA2 couldn't finish in limited time (>30 Days).

2.4 Analyses on UK Biobank data

To illustrate how it can be useful to perform PCA on large genetic datasets, we performed PCA analyses on UK Biobank imputed data with 487,409 individuals and 6,133,304 SNPs without LD pruning in order to capture the local and global genetic patterns. We investigated the distribution of the PC loadings. As shown in Figure S6 and Figure 3, most PCs capture low recombination regions such as centromeres and inversions. However, PC1, PC2 and PC4 capture population structure with their highest SNP loadings located in genes known to be under selection, such as PC1 and PC2 with a peak in the skin pigmentation gene *SLC45A2* (Beleza et al. 2013) and PC4 with a peak in the KRAB-ZNF cluster (Perdomo-Sabogal et al. 2019). Other interesting peaks include the large inversion found on chromosome 8 (Porubsky et al. 2022; Bansal et al. 2007) and the HLA region on chromosome 6.

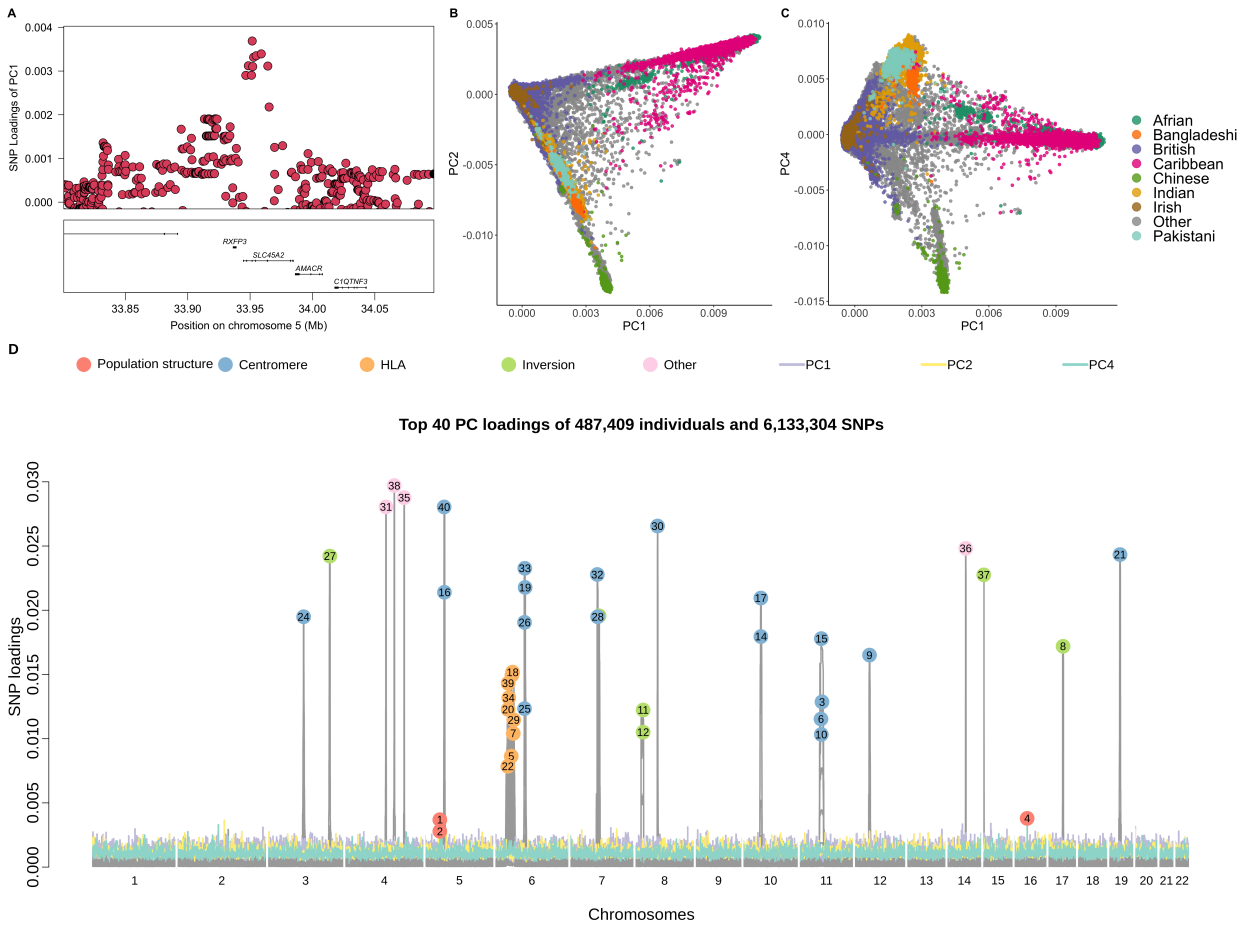

Figure 3: PCA of the whole UK Biobank imputed dataset. **A**, Locuszoom of variant rs16891982 with highest loadings for PC1 and PC2. Population structure showed in PC1 vs PC2 **B** and PC1 vs PC4 **C** for all 487,409 individuals, colored by country or region label. Other labels are merged into the category "other". **D**, SNP loadings for the top 40 PCs. Peaks are annotated and colored by the type of genomic region they are located in.

2.5 Analyses on single cell data

In the PCAone software, we have implemented not only multiple PCA algorithms but also supporting different data types. To show the application of PCAone in the field of single cell RNA sequencing (scRNAs), we performed PCA on one of the largest single cell RNA sequence data with 1,306,127 mice brain cells and 23,771 genes, which took 71 and 47 minutes for PCAone and PCAone$_{H+Y}$ respectively to calculate top 40 PCs while using little memory. We compared the per-

formance of PCAone with the OnlinePCA.jl-Halko method which a recent study showed was the fastest out-of-core implementation available for single cell RNA sequencing datasets (Tsuyuzaki et al. 2020). As the file format of storing the input matrix can have a huge difference on the performance of out-of-core implementations, a binary file would be the ideal format for fast I/O access, which is used by OnlinePCA.jl. However, the CSV format is more widely used compared to a customized binary file. PCAone supports directly the general CSV format as input. By default PCAone assumes the data is normalized by the users so that they can apply their own normalization method. But PCAone also supports the common normalization method for scRNAs (see Methods). Notably, OnlinePCA.jl splits the calculation of normalization and PCA into two processes while PCAone performs file parsing, normalization and PCA at the same time. We evaluated the time of IO, normalization and PCA computing separately for each program. As Table 2 shown, PCAone consumed less I/O and computing time than OnlinePCA.jl leveraging 20 threads. Even though PCAone uses a slower to parse but general CSV format as input, we showed that PCAone is more then 5x times faster than OnlinePCA.jl using 20 threads. As full SVD is not computationally feasible and we therefore used the results from the IRAM method PCAone$_{Arnoldi}$ when calculating MEV and minSSE. To illustrate that the IRAM implementation is an appropriate substitute for the full SVD, we also performed the analysis on a subset of 12000 cells where the accuracy is the same when we substitute the full SVD for PCAone$_{Arnoldi}$ (Table S3). As shown in Figure 4, OnlinePCA$_{Halko}$ has lower accuracy for tailing PCs when calculating the top 40 PCs. Both PCAone and PCAone$_{H+Y}$ have high accuracy and as also show in Figure 4 the PCA from PCAone is visually indistinguishable from the results from the slower IRAM method PCAone$_{Arnoldi}$ which is not the case for OnlinePCA$_{Halko}$.

Table 2: Performance on scRNAs data with 1,306,127 cells and 23,771 genes for estimating the top K=40 PCs. Time is given in minutes and RAM in gigabytes. Wall time includes IO time. * normalization of data is performed in a separate step by OnlinePCA.jl while PCAone does in on the fly. For wall time, we have not included the time needed by OnlinePCA$_{Halko}$ to make their binary format used for quick IO or the time of permutation potentially needed for PCAone. PCs estimated by PCAone$_{Arnoldi}$ are used as the true value for calculating accuracy (MEV and minSSE). All programs used 20 CPU threads.

Program	Wall Time(m)	IO Time(m)	Epochs	RAM(GB)	MEV	minSSE
PCAone	71	54	10	8.97	0.999957	0.004171
PCAone$_{H+Y}$	53	43	8	6.82	0.999925	0.007328
OnlinePCA$_{Halko}$	461+100*	300	8	1.71	0.954103	7.102932
PCAone$_{Arnoldi}$	484	469	103	5.73	-	-

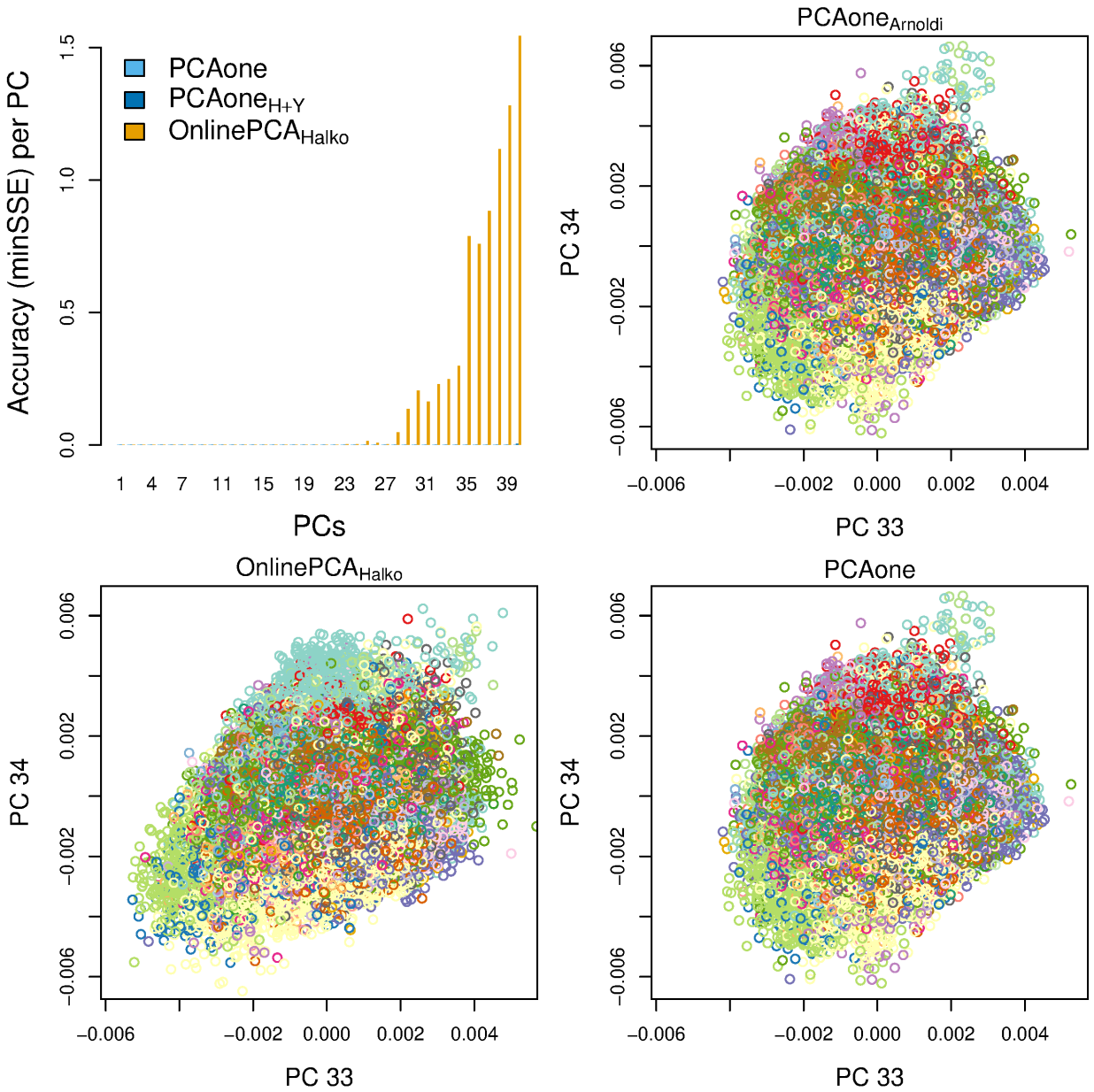

Figure 4: Accuracy of individual PCs for scRNAs data with 1,306,127 cells and 23,771 genes. The top left plot is the accuracy (minSSE) of each PC where PCAone$_{Arnoldi}$ is used as the true value. PC 33 and 34 are shown for PCAone and OnlinePCA$_{Halko}$ to illustrate the difference compared to the accurate PCAone$_{Arnoldi}$.

3 Discussion

We have presented PCAone, an out-of-core PCA framework designed for large-scale data analysis. Based on multiple datasets we have demonstrated that our IRAM implementation, PCAone$_{Arnoldi}$, is much faster than the other accurate IRAM method FlashPCA2 used in genetics, as well as being the only out-of-core implementation of IRAM supporting scRNAs data. It is faster mainly due to being multithreaded and it allows us to benchmark the different RSVD methods against it when the data is too large for the full SVD method. In all analysis our new window-based RSVD algorithm, PCAone, has high accuracy and is much faster than existing RSVD algorithms. This is especially true when the eigenvalues of the estimated PCs decay slowly, where PCAone needs far fewer number of epochs to converge, thus making it much faster than other normal RSVD

methods. We have demonstrated this point using genotype data from the genetically similar East Asian populations of the 1000 Genomes Project, where the number of epochs needed for convergence is much higher for the other methods while PCAone only needs 7. For this dataset the accuracy of different methods compared against the full-rank SVD varied a lot. Unsurprisingly the IRAM methods, PCAone$_{Arnoldi}$ and FlashPCA2, performed the best but the much faster PCAone algorithm performed almost as well and produced PCs that were visually indistinguishable from the full SVD. The other algorithms based on RSVD performed much worse for the East Asian dataset. The performance of Plink2(FastPCA) and ProPCA increased when many PCs were estimated which is not surprising since the number of PCs determines their number of optimization steps. Most of the methods can be run with user defined stopping criteria which improve their general performance at the expense of speed. PCAone often uses exactly 7 epochs given the default setting of using 64 blocks for it expanding window optimization scheme(see Figure S2). This is despite having a stopping criteria based on the difference in PCs between consecutive epochs. This is mainly due to the optimization steps of the algorithm which uses data from expanding window sizes where epoch 6 and 7 are the first consecutive iterations that use all of the data. The reason PCAone converges in fewer epochs than the other RSVD methods is that it performs updates of the initial random matrix Ω many times in the first 5 epochs. For example in the first epoch Ω is updated 64 times based on random subsets of the data. This is a very efficient approach when many of the features are correlated with many other features which is the case for dense genetic data. Therefore, performing the optimization on subset of the data will approximate the solution from the full data set. We increase the size of the subsets in each epoch until the full data is used in epoch 6 which is the same as a Halko power iteration update of Ω. Therefore, the algorithm will still work even if the features are not correlated.

We benchmarked the performance of existing PCA methods in terms of speed and memory for both genetic data and scRNAs data. Based on different subsets of the imputed genotype data of the UK Biobank, PCAone is the fastest algorithm followed by the other RSVD methods, PCAone$_{H+Y}$, Plink2, ProPCA and TeraPCA. Plink2(FastPCA) and ProPCA are not out-of-core implementations and therefore should have a great advantage in terms of speed. However, they use a large amount of memory and are therefore not applicable to very large datasets. Surprisingly our out-of-core RSVD implementation PCAone and PCAone$_{H+Y}$ are much faster than Plink2 and ProPCA despite having I/O operations being much more time consuming than computations. Likewise for scRNAs, PCAone and PCAone$_{H+Y}$ are much faster than OnlinePCA$_{Halko}$ even though the latter uses a more efficient input format. The main advantage of PCAone is that it converges in less epochs compared to other methods such as Plink2(FastPCA) and ProPCA. However, the implementation of PCAone has a faster and more efficent multithreaded implementation which also is used when reading data from disk. In theory, the I/O performance of disk matters a lot as currently the speed of disk I/O should be always slower than CPU. Ideally, the faster the disk is, the faster the out-of-core algorithm is.

To illustrate how PCAone can be useful for biological researchers, we analyzed all of the 6 million common imputed variants in the UK Biobank consisting of almost half a million individuals, which only took 9 hours using 20 GBs memory. This allowed us to explore the population structure as well as the correlation between markers as we did not perform LD pruning. Therefore, we capture both the local and global genetic patterns, which has not been explored in previous studies due to the limitations in existing PCA methods. As a result, most PCs capture regions with low recombination rates such as centromeres while PC1, PC2, PC4 capture the population structure. For both PC1 and PC2, the highest SNP loadings are located in the known skin pigmentation gene *SLC45A2*. Other interesting peaks include the large inversion found in chromosome 8 and the HLA region on chromosome 6. Most of these regions are reported as long range LD regions in previous studies (Price, Weale, et al. 2008; Bycroft et al. 2018). We also performed PCA on scRNAs data with 1,306,127 mice brain cells and 23,771 genes, which demonstrated that PCAone can be 10x faster than existing out-of-core PCA implementation OnlinePCA.jl-Halko in scRNAs

field while being more accurate. PCAone takes a general CSV file as input instead of a customized binary format as used by OnlinePCA.jl, which makes PCAone easier to use.

Beyond the three fundamental PCA algorithms, one can take advantage of the PCAone framework to implement other PCA related methods. For instance, we can deal with missingness in the genotypes, using genotype dosages or genotype likelihoods, which is achieved with an implementation of the PCAngsd method (Meisner and Albrechtsen 2018) for genotype likelihoods and the EMU algorithm (Meisner, Liu, et al. 2021) for rampant missingness including having no data overlap between many pairs of samples. While the out-of-core algorithms can always cope with limited memory issues whenever the RAM usage is large, the I/O operations of parsing the input can be a bottleneck of the performance. Therefore, faster file parsing and more pass-efficient algorithm are both essential to maximize the performance. The PCAone framework is highly optimized for out-of-core computations and implemented as a multithreaded C++ software. Another way to solve big memory issue is using memory-mapped file such as bigsnpr (Privé, Aschard, et al. 2018). And we also implemented PCAone algorithms as a R package such that one can easily use PCAone with the existing memory-mapped object in R, eg. bigmemory (Kane et al. 2013).

4 Methods

4.1 PCAone algorithms

Our proposed algorithm is based on the Halko et al. 2011 randomized SVD. Therefore we briefly explain the underlying idea for this algorithm, which seeks a near-optimal orthonormal basis $Q^{m\times k}$ for input matrix $A^{m\times n}$ such that $A \approx QQ^TA$ with $k \ll m, n$. Here n denotes the number of samples and m denotes the number of features. With such a Q, the right eigenvectors of A are approximated by the right eigenvectors of the low dimensional $B = Q^TA$. Halko et al. 2011 proposed to find the basis Q by multiplying the input matrix with a random low-dimensional matrix $\Omega^{n\times k}$ drawn from a standard normal distribution and then performing QR factorization $(Q, R = qr(A\Omega))$ on this product. This is the RSVD algorithm which can be performed on batches of the data so the analyses can be updates with new data, but it goes though the data twice. Recently, Yu et al. 2017 proposed a more efficient version of this algorithm that goes through the input matrix only once and includes re-orthogonalization of Q which alleviates round of errors. However, these single pass algorithm is not always very accurate and Halko et al. 2011 also proposed an more accurate algorithm. To find a Q that can yield a better approximation Halko et al. 2011 proposed to optimize Ω by another orthogonalization process $\Omega, R = qr(A^TQ)$. Those two computation steps are the so-called power iteration steps.

As noted by (Yu et al. 2017) it is possible to add the power iteration scheme from (Halko et al. 2011) to their their algorithm. We have implemented such an algorithm (Algorithm 1) which we denote PCAone_{H+Y}. Note that when we want to estimate k PCs we will use a matrix of near optimal bases Q that has $l = k + 10$ columns. This padding of 10 extra PCs alleviates some of the issues of eigenvalues switching order in the optimization.

Algorithm 1 PCAone$_{H+Y}$: RSVD algorithm based on Yu et al. 2017 but with power iterations

Given: Input matrix $A \in \mathbb{R}^{m \times n}$, p iterations, target ranks k, oversampling ranks 10

Divide A into b batches by rows $A = [A_1, A_2, \ldots, A_b]$, set b according to desired memory usage
Draw random matrix $\Omega^{(0)} \in \mathbb{R}^{n \times l}$ from a standard Guassian distribution
for $i = 1 : p$ **do**
    set H to a $n \times l$ zero matrix and set $G = []$ to a empty matrix
    for $j = 1 : b$ **do**
        $g \leftarrow A_b \Omega^{(i-1)}$
        $G \leftarrow [G, g]$ ▷ append g to G matrix
        $H \leftarrow H + A_j^T g$
    end for
    $\Omega^{(i)}, R^{(i)} \leftarrow qr(H^{(i)})$ ▷ update Ω (power iteration step)
end for
$Q_1, R_1 \leftarrow qr(G^{(p)})$
$Q_2, R_2 \leftarrow qr(Q_1)$
$R \leftarrow R_1 R_2$ ▷ re-orthogonalization step
$B \leftarrow R^{-T} H$
$\hat{U}, S, V \leftarrow svd(B)$
$U \leftarrow Q\hat{U}$

Return: $U_{1:k}, S_{1:k}, V_{1:k}$

The algorithm of Yu et al. 2017 allows us to only read in the data in small batches once per iteration, which makes it very useful for large datasets that does not fit in memory and by adding power iterations to it, we can gain higher accuracy. For each power iteration the data is read into memory from the disk which is an issue for datasets where many power iterations are needed in order to achieve accurate top PCs. We denote the number of times the data must be read from the disk as epochs and in order to reduce the number of epochs we suggest an algorithm that allows for multiple updates of Ω (power iterations) in each epoch. The algorithm denoted as PCAone is seen here Algorithm 2. First we permute the input matrix A by rows (features) and divide it into 64 blocks of approximately equal size. In this way each block of the data is a random subset of the features. This is important for datasets where the adjacent features are correlated. For example genetic data where SNPs are stored according to position and are locally correlated due to linkage disequilibrium. In the first epoch we perform power iterations based on a window of data. We start with a window size of 2 blocks of data and from this subset of data we perform the power iterations which update Ω. We then slide the window one block so that the next iteration which updates Ω is performed on block 2 and 3. Although we use data from 2 block we don't need to keep the data in the 2 blocks in memory because we can construct the H and G matrices from smaller batches of data as in Algorithm 1. However, because updates of Ω are based on a QR factorization, the sign of each of its column will change in each iteration. We therefore match them between iterations by flipping the sign if needed.

After reading through the whole data (one epoch), we double both the step size and the window size until epoch number 6 where the window size equals the whole input matrix as illustrated in S2. Thereafter, the power iterations are performed in the same way as in the normal RSVD. The algorithm stops when the difference of eigenvectors between two successive epochs is smaller than an user defined threshold (default 10^{-4}). This often happens after 6 epochs such where a second update of Ω based on the whole data only leads to a very small change of Ω and thus the estimated PCs. For simplicity, the PCAone algorithm shown in Algorithm 2 as if the whole block of data is read into memory, however for large datasets, we only keep small batches of the data matrix A in memory as shown in Algorithm 1.

Algorithm 2 PCAone: RSVD with window based power iterations. The sliding window approach is illustrated in S2

Given: Input matrix $A \in \mathbb{R}^{m \times n}$, e epochs, target ranks k, oversampling ranks 10

1: Permute the rows of A and divide into $s = 2^6 = 64$ blocks by rows $A = [A_1, A_2, \ldots, A_s]$
2: Draw random matrix $\Omega^{(0)} \in \mathbb{R}^{n \times l}$ from a standard Gaussian distribution
3: $\gamma \leftarrow 1; i \leftarrow 1$ ▷ initial step size $\gamma = 1$ (window size 2γ)
4: **for** $e = 1, 2, ...$ **do**
5: **for** $j = 1 : s$ **do**
6: $g_j \leftarrow A_j \Omega^{(i-1)}; h_j \leftarrow A_j^T g_j$ ▷ done in batches (see Algorithm 1)
7: **if** j is last block of the window **then** ▷ $j \mod \gamma == 0$ & $2j \neq min(2^e, s)$
8: $H = \sum_{l=j-2\gamma+1}^{j} h_{f(l)}$ ▷ $f(l) = (l - 1 \mod s) + 1$
9: $\Omega^{(i)}, R^{(i)} \leftarrow qr(H)$ ▷ flip signs of $\Omega^{(i)}$ to match $\Omega^{(i-1)}$
10: $i \leftarrow i + 1$
11: $2\gamma \leftarrow min(2^e, s)$ ▷ epoch determines step(γ) and window size(2γ)
12: **end if**
13: **end for**
14: **end for**
15: $G = [g_1, g_2, ..., g_s]$
16: use step 12–17 in Algorithm 1 to obtain top k eigenvectors from H and G.

Return: $U_{1:k}, S_{1:k}, V_{1:k}$

The major difference between PCAone$_{H+Y}$ (Algorithm 1) and PCAone (Algorithm 2) is the number of power iterations performed each epoch. In the five first epochs PCAone updates Ω 124 times while PCAone$_{H+Y}$ only does it five times. Starting from the 6th epoch, the algorithms are the same and Ω is updated only once per epoch.

4.2 Convergence and accuracy measurement

To measure the similarity of two eigenvector matrices, we use the mean explained variance (MEV) (Galinsky et al. 2016; Agrawal et al. 2020) and minSSE. Both measures are indifferent to the matrix's order of the eigenvectors. Assuming we have a true set of PCs $(v_1, v_2, ..., v_k)$, we evaluate the accuracy of a set of estimated PCs $(u_1, u_2, ..., u_k)$ as follows:

$$\text{MEV} = \frac{1}{m} \sum_{i=1}^{k} \sqrt{\sum_{i}^{k} (v_j \cdot u_i)^2} = \frac{1}{m} \sum_{i=1}^{k} \|U^T v_i\|,$$

where U is a matrix with the estimated PCs as column vectors.

MEV can become very small for large data sets even if there are some substantial differences between a few values. Therefore we also use a measure based on the sum of squared error between the two matrices that allows for reordering of the columns. We call this measure minSSE defined as

$$\text{minSSE} = \sum_{i=1}^{k} min_j \|v_j - u_i\|.$$

The minSSE is a measure of the minimum squared distance between matrix U to a matrix based on a combination of v_j.

For some RSVD implementations, the users have to specify the number of power iterations themselves, which therefore lacks an explicit measure of accuracy. For PCAone$_{H+Y}$ and PCAone we can obtain the eigenvectors and eigenvalues after each power iteration, such that we can easily compare two sets of estimated PCs from successive power iterations and implement an explicit

stopping criteria. We stop performing power iterations when $1 - \text{MEV} \leq 10^{-4}$ is met. The threshold can also be specified by the user.

4.3 Datasets and benchmarking

To benchmark the performance of all PCA methods, we use multiple real-world datasets. First, we downloaded the whole-genome sequencing genotype data with GRCh38 coordinates from the 1000 genomes project (Byrska-Bishop et al. 2021). In one analysis we used 400 individuals from four Asian populations: two Han Chinese population (CHB, CHS), one Dai Chinese (CDX) and one Vietnamese (KHV) population. We used 2 millions random SNPs with minor allele frequency MAF > 0.05. We obtained imputed genotype data from the UK Biobank (Bycroft et al. 2018). We only performed filtering on the MAF and included SNPs with a MAF > 0.05, such that 6,133,304 SNPs were retained. For benchmarking the speed, memory usage and scalability of different methods, we created multiple datasets by randomly sampling individuals and SNPs from the UK Biobank data. Additional to genetic data, we downloaded the public single cell RNA sequencing (scRNAs) data with raw gene expression counts from `https://support.10xgenomics.com/single-cell-gene-expression/datasets/1.3.0/1M_neurons` to benchmark PCAone's performance on scRNAs data, which consists of 1,306,127 mice brain cells from 60 clusters and and 23,771 genes generated with the 10X Chromium (Zheng et al. 2017). The label of each cell belonging to which cluster was used to color the cell point in PCA plots. We also created another small dataset with 200 cells each cluster and 23,771 genes by randomly sampling cells from each cluster of the big dataset, for which we can have the PCs computed by full-rank SVD. All benchmarking were performed on a cluster with 192 GB of memory and 40 CPU threads available (CPU model: Intel(R) Xeon(R) Gold 6230 CPU @ 2.10GHz).

4.4 Normalization for genetic data

For inferring population structure with genetic data, PCAone supports genotypes (Plink binary format) or genotype dosage (Bgen format) as input. Let G be a $m \times n$ genotype matrix from n individuals at m SNPs. The genotype vector for individual j is a vector of length m denoted by $g_j \in [0, 2]^m$ where each entry $g_{i,j} \in [0, 2]$ denotes the number of minor alleles carried by individual j at SNP i or the genotype dosage. Let A denote the matrix of standardized genotypes where each row a_i has approximately mean 0 and variance 1 for SNPs in Hardy-Weinberg equilibrium.

$$\hat{f}_i = \frac{1}{2n} \sum_{j}^{n} g_{i,j}$$

$$a_{i,j} = \frac{g_{i,j} - 2\hat{f}_i}{\sqrt{2\hat{f}_i(1 - \hat{f}_i)}}$$

4.5 Normalization for single-cell RNA sequencing data

PCAone supports a general csv format as input where each row indicates the samples for the feature. By default, PCAone assumes the data in CSV format is already normalized so that users can apply their own normalization methods. We also implemented count per median (CPMED) normalization method for single cell RNAs data in PCAone (with option *–cpmed*), which is the commonly used one (Shekhar et al. 2016; van Dijk et al. 2018) and the one implemented by OnlinePCA.jl (Tsuyuzaki et al. 2020), so that we can benchmark the performance of PCAone and OnlinePCA.jl with the same normalization method. Let X be the raw matrix with m genes and n cells where each row x_i is a vector of length n and each entry $x_{i,j}$ denotes the read counts or TPM for cell i at gene j. Let $s = [s_1, s_2, ..., s_n]$ be a vector where each entry is sum of each

column of X. Let A denote the normalized matrix where each entry $a_{i,j}$ is median-normalized and log-transformed as the follows.

$$s_j = \sum_{i}^{m} x_{i,j}$$

$$a_{i,j} = \log_{10}\left(\frac{x_{i,j}}{s_j} \times median(s) + 1\right)$$

5 Availability of data and materials

The reproducible workflow used to perform the analyses in the paper is available at `https://github.com/Zilong-Li/Papers/tree/main/pcaone`. R package in on CRAN `https://CRAN.R-project.org/package=pcaone`. The mice Brain scRNAs data is downloaded from `https://support.10xgenomics.com/single-cell-gene-expression/datasets/1.3.0/1M_neurons`. The 1000 genome project data is downloaded from `http://ftp.1000genomes.ebi.ac.uk/vol1/ftp/data_collections/1000G_2504_high_coverage/working/20201028_3202_raw_GT_with_annot/`. The UK Biobank data is granted under Application Number 32683 from the UK Biobank Resource.

6 Acknowledgements

The study was supported by the Lundbeck foundation (R215-2015-4174) and the Novo Nordisk Foundation (NNF20OC0061343). This research has been conducted using the UK Biobank Resource under Application Number 32683. None of the authors declared conflict of interest.

References

Abraham, G, Y Qiu, and M Inouye (2017). "FlashPCA2: principal component analysis of Biobank-scale genotype datasets". In: *Bioinformatics (Oxford, England)* 33.17, pp. 2776–2778. DOI: 10.1093/bioinformatics/btx299.

Agrawal, A, AM Chiu, M Le, E Halperin, and S Sankararaman (2020). "Scalable probabilistic PCA for large-scale genetic variation data". In: *PLOS Genetics* 16.5, e1008773. DOI: 10.1371/journal.pgen.1008773.

Bansal, V, A Bashir, and V Bafna (2007). "Evidence for large inversion polymorphisms in the human genome from HapMap data". In: *Genome Research* 17.2, pp. 219–230. DOI: 10.1101/gr.5774507.

Beleza, S, AM Santos, B McEvoy, I Alves, C Martinho, E Cameron, MD Shriver, EJ Parra, and J Rocha (2013). "The Timing of Pigmentation Lightening in Europeans". In: *Molecular Biology and Evolution* 30.1, pp. 24–35. DOI: 10.1093/molbev/mss207.

Bose, A, V Kalantzis, EM Kontopoulou, M Elkady, P Paschou, and P Drineas (2019). "TeraPCA: a fast and scalable software package to study genetic variation in tera-scale genotypes". In: *Bioinformatics* 35.19, pp. 3679–3683. DOI: 10.1093/bioinformatics/btz157.

Bycroft, C et al. (2018). "The UK Biobank resource with deep phenotyping and genomic data". In: *Nature* 562.7726, pp. 203–209. DOI: 10.1038/s41586-018-0579-z.

Byrska-Bishop, M et al. (2021). "High coverage whole genome sequencing of the expanded 1000 Genomes Project cohort including 602 trios". In: *bioRxiv*, p. 2021.02.06.430068. DOI: 10.1101/2021.02.06.430068.

Chang, CC, CC Chow, LC Tellier, S Vattikuti, SM Purcell, and JJ Lee (2015). "Second-generation PLINK: rising to the challenge of larger and richer datasets". In: *GigaScience* 4.1, p. 7. DOI: 10.1186/s13742-015-0047-8.

Diaz-Papkovich, A, L Anderson-Trocmé, C Ben-Eghan, and S Gravel (2019). "UMAP reveals cryptic population structure and phenotype heterogeneity in large genomic cohorts". In: *PLOS Genetics* 15.11, e1008432. DOI: 10.1371/journal.pgen.1008432.

Galinsky, KJ, G Bhatia, PR Loh, S Georgiev, S Mukherjee, NJ Patterson, and AL Price (2016). "Fast Principal-Component Analysis Reveals Convergent Evolution of ADH1B in Europe and East Asia". In: *The American Journal of Human Genetics* 98.3, pp. 456–472. DOI: 10.1016/j.ajhg.2015.12.022.

Halko, N, PG Martinsson, and JA Tropp (2011). "Finding Structure with Randomness: Probabilistic Algorithms for Constructing Approximate Matrix Decompositions". In: *SIAM Review* 53.2, pp. 217–288. DOI: 10.1137/090771806.

Kane, MJ, J Emerson, and S Weston (2013). "Scalable Strategies for Computing with Massive Data". In: *Journal of Statistical Software* 55.14, pp. 1–19.

Kiselev, VY, TS Andrews, and M Hemberg (2019). "Challenges in unsupervised clustering of single-cell RNA-seq data". In: *Nature Reviews Genetics* 20.5, pp. 273–282. DOI: 10.1038/s41576-018-0088-9.

Lehoucq, RB and DC Sorensen (1996). "Deflation Techniques for an Implicitly Restarted Arnoldi Iteration". In: *SIAM Journal on Matrix Analysis and Applications* 17.4, pp. 789–821. DOI: 10.1137/S0895479895281484.

Lehoucq, RB, DC Sorensen, and C Yang (1998). *ARPACK Users' Guide: Solution of Large-Scale Eigenvalue Problems with Implicitly Restarted Arnoldi Methods*. Society for Industrial et al. DOI: 10.1137/1.9780898719628.

Maaten, L van der and G Hinton (2008). "Visualizing Data using t-SNE". In: *Journal of Machine Learning Research* 9.86, pp. 2579–2605.

Meisner, J and A Albrechtsen (2018). "Inferring Population Structure and Admixture Proportions in Low-Depth NGS Data". In: *Genetics* 210.2, pp. 719–731. DOI: 10.1534/genetics.118.301336.

Meisner, J, S Liu, M Huang, and A Albrechtsen (2021). "Large-scale inference of population structure in presence of missingness using PCA". In: *Bioinformatics* 37.13, pp. 1868–1875. DOI: 10.1093/bioinformatics/btab027.

Patterson, NJ, AL Price, and D Reich (2006). "Population Structure and Eigenanalysis". In: *PLOS Genetics* 2.12, e190. DOI: 10.1371/journal.pgen.0020190.

Perdomo-Sabogal, Á and K Nowick (2019). "Genetic Variation in Human Gene Regulatory Factors Uncovers Regulatory Roles in Local Adaptation and Disease". In: *Genome Biol Evol* 11.8, pp. 2178–2193.

Porubsky, D et al. (2022). "Recurrent inversion polymorphisms in humans associate with genetic instability and genomic disorders". In: *Cell* 0.0. DOI: 10.1016/j.cell.2022.04.017.

Price, AL, NJ Patterson, RM Plenge, ME Weinblatt, NA Shadick, and D Reich (2006). "Principal components analysis corrects for stratification in genome-wide association studies". In: *Nature Genetics* 38.8, pp. 904–909. DOI: 10.1038/ng1847.

Price, AL, ME Weale, et al. (2008). "Long-Range LD Can Confound Genome Scans in Admixed Populations". In: *The American Journal of Human Genetics* 83.1, pp. 132–135. DOI: 10.1016/j.ajhg.2008.06.005.

Privé, F, H Aschard, A Ziyatdinov, and MGB Blum (2018). "Efficient analysis of large-scale genome-wide data with two R packages: bigstatsr and bigsnpr". In: *Bioinformatics* 34.16, pp. 2781–2787. DOI: 10.1093/bioinformatics/bty185.

Privé, F, K Luu, MGB Blum, JJ McGrath, and BJ Vilhjálmsson (2020). "Efficient toolkit implementing best practices for principal component analysis of population genetic data". In: *Bioinformatics* 36.16, pp. 4449–4457. DOI: 10.1093/bioinformatics/btaa520.

Shekhar, K et al. (2016). "Comprehensive Classification of Retinal Bipolar Neurons by Single-Cell Transcriptomics". In: *Cell* 166.5, 1308–1323.e30. DOI: 10.1016/j.cell.2016.07.054.

Tsuyuzaki, K, H Sato, K Sato, and I Nikaido (2020). "Benchmarking principal component analysis for large-scale single-cell RNA-sequencing". In: *Genome Biology* 21.1, p. 9. DOI: 10.1186/s13059-019-1900-3.

van Dijk, D et al. (2018). "Recovering Gene Interactions from Single-Cell Data Using Data Diffusion". In: *Cell* 174.3, 716–729.e27. DOI: 10.1016/j.cell.2018.05.061.

Yu, W, Y Gu, J Li, S Liu, and Y Li (2017). "Single-Pass PCA of Large High-Dimensional Data". In: pp. 3350–3356.

Zheng, GXY et al. (2017). "Massively parallel digital transcriptional profiling of single cells". In: *Nature Communications* 8, p. 14049. DOI: 10.1038/ncomms14049.

Supplementary Material

S1 Supplementary figures

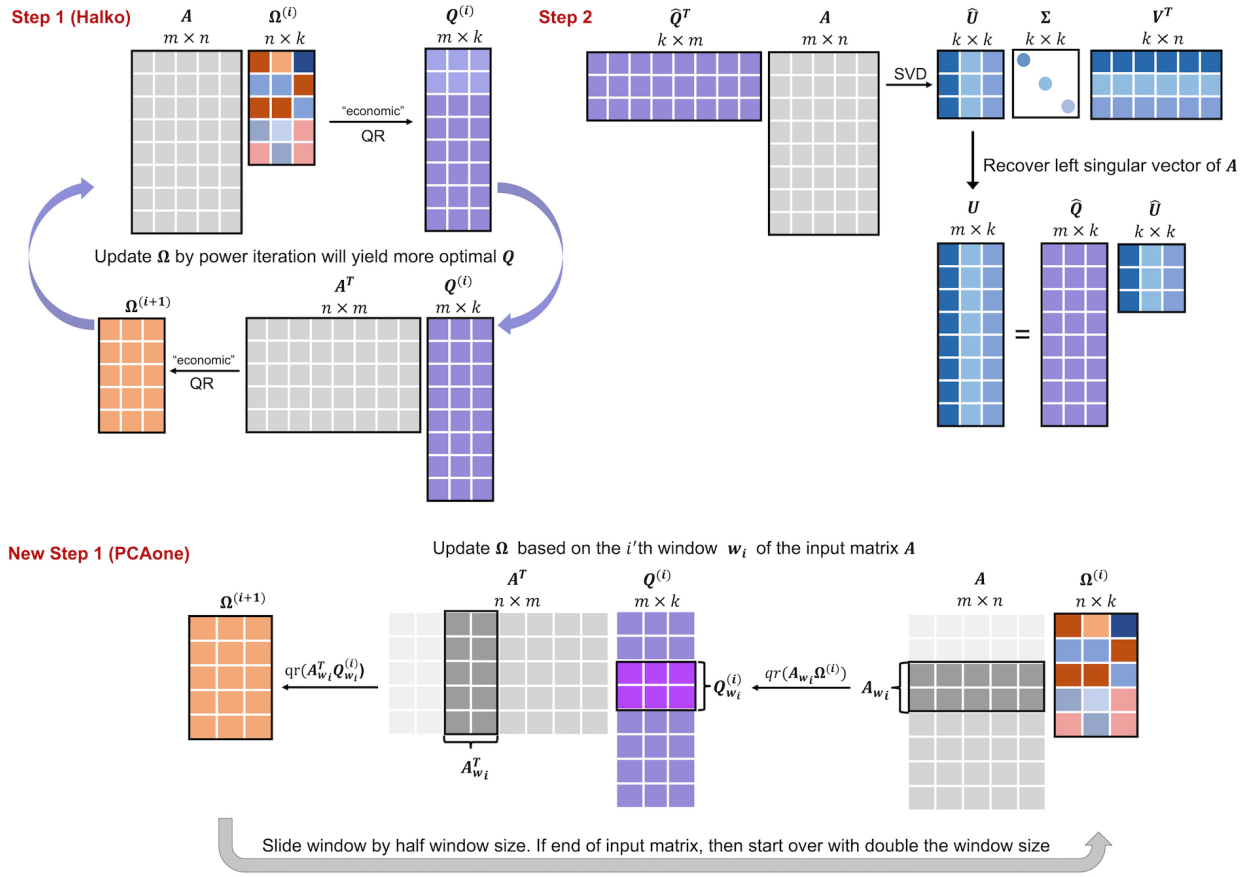

Figure S1: Overview of PCAone algorithm. The underlying idea of Halko randomized SVD is seeking a near-optimal orthonormal basis $Q^{m\times k}$ for input matrix $A^{m\times n}$, such that $A \approx QQ^TA$ is satisfied with $k \ll m, n$. In **step1 of the Halko**, Q is obtained from QR orthogonalization $Q = qr(A\Omega)$ of a random low-dimensional matrix $\Omega^{n\times k}$ multiplied with the input matrix. Better accuracy is achived by iterative updating Q and Ω, which composes a so-called power iteration step. In **Step2**, with such a Q the right eigenvectors of A are approximated by the right eigenvectors of the low dimensional Q^TA. For large datasets, a **new step1 in PCAone** is proposed, which uses a subset of the input matrix A_{w_i} each time to perform power iterations in a sliding window with step size of half the window. After each pass through the whole data, the window size of A_{w_i} is doubled until A_{w_i} equals A.

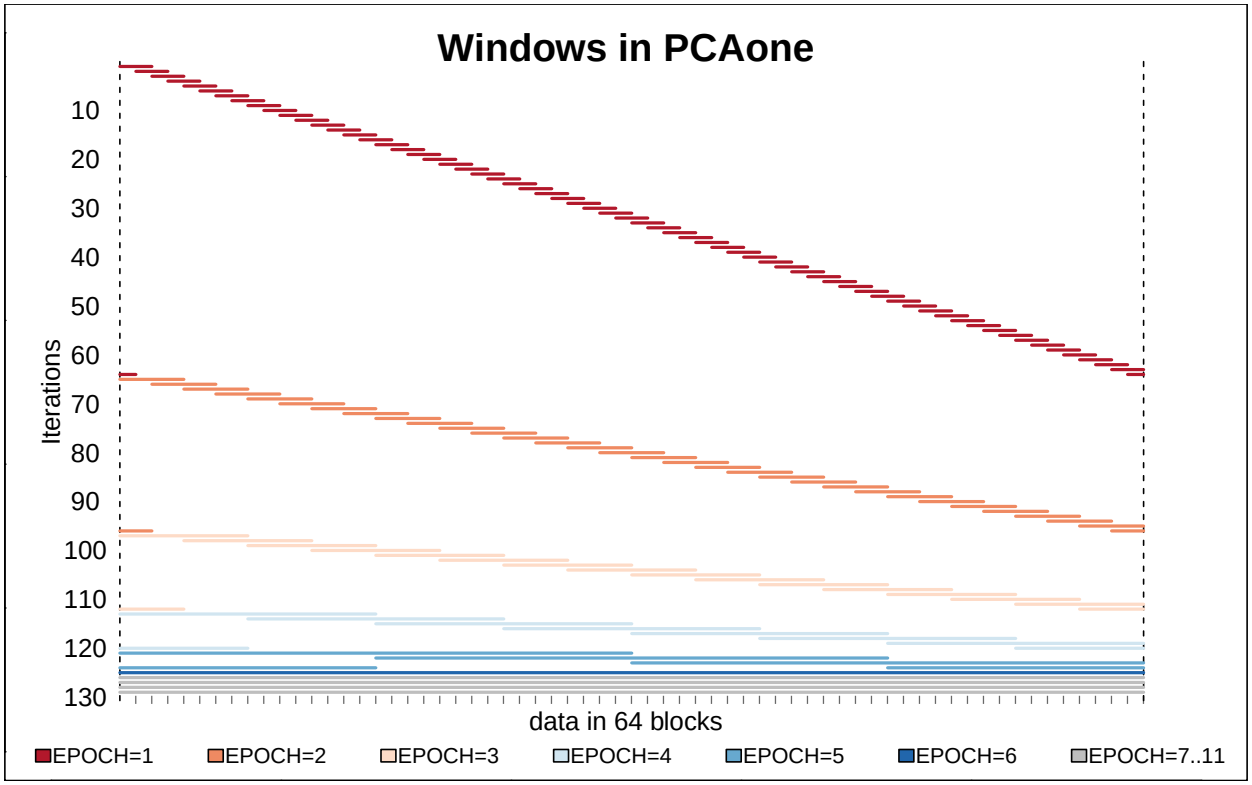

Figure S2: Illustration of the windows of data used for each iteration where each horizontal line represents parts of the data that is used to update Ω at each iteration. Each epoch denotes the reading of the data from the disk and each iteration is colored by its epochs. In this illustration the data is split into 64 blocks and in the first epoch a window size of 2 and a step size of 1 is used. The window wraps around the data such as the window starting at the last block will use data from the first 1 blocks as well. In each of the following epochs both the window size and the step size is doubled. For the first 6 epocs the data is read 6 times from the disk but Ω is updated **64+32+16+8+4+1=124** times where the last iteration uses the whole data.

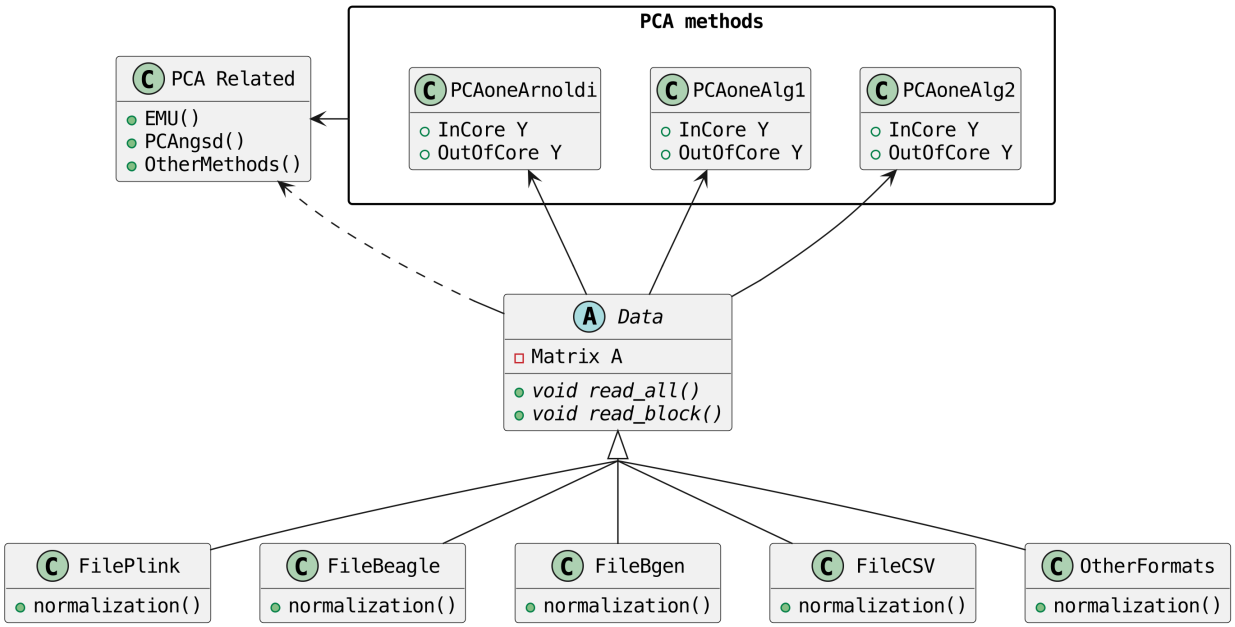

Figure S3: Overview of PCAone Architecture. Multiple file formats for diallelic genotype data (Plink binary), genotype likelihoods (beagle imputation format), allelic dosages from imputation (Bgen), count data such as scRNA (CSV) or a simple matrix of real values (CSV). PCAone also works with missing data using EMU algorithm (Meisner, Liu, et al. 2021) or for genotype likelihoods from low depth sequencing data using PCAngsd algorithm (Meisner and Albrechtsen 2018).

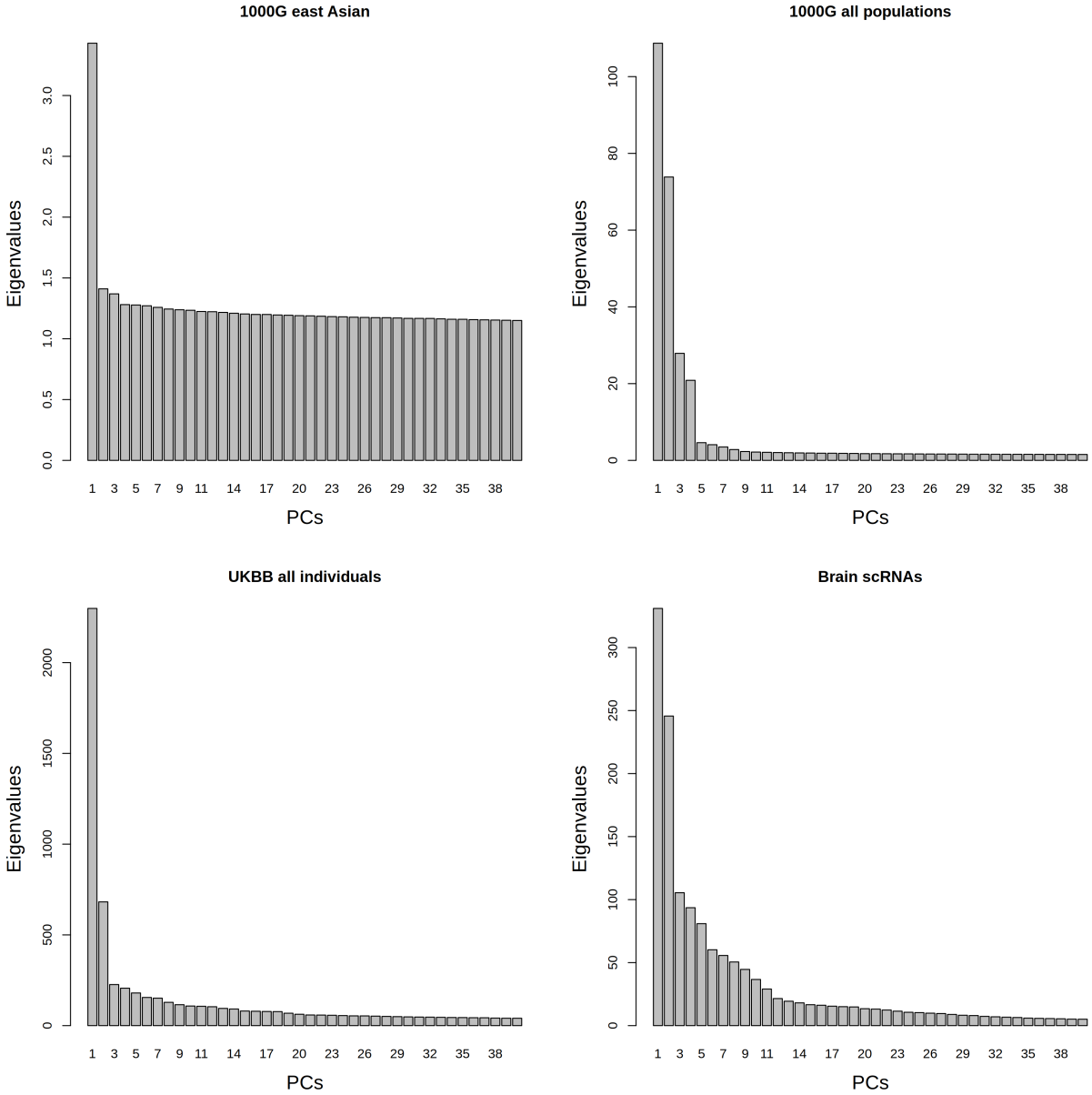

Figure S4: Top 40 eigenvalues for 4 different datasets. For 1000 genomes, the results are from full SVD. For UKBB and scRNAs, the results are from PCAone$_{Arnoldi}$

A Number of passes over data (epochs) of each method for top K PCs

K(PCs)	PCAone	PCAone$_{H+Y}$	TeraPCA	Plink2(FastPCA)*	ProPCA*	FlashPCA2	PCAone$_{Arnoldi}$
2	2	3	6	2	4	10	21
5	4	6	12	5	7	25	21
10	7	11	22	10	12	55	46
20	7	18	42	20	22	117	98
30	7	19	30	30	32	172	144
40	7	19	36	40	42	223	188

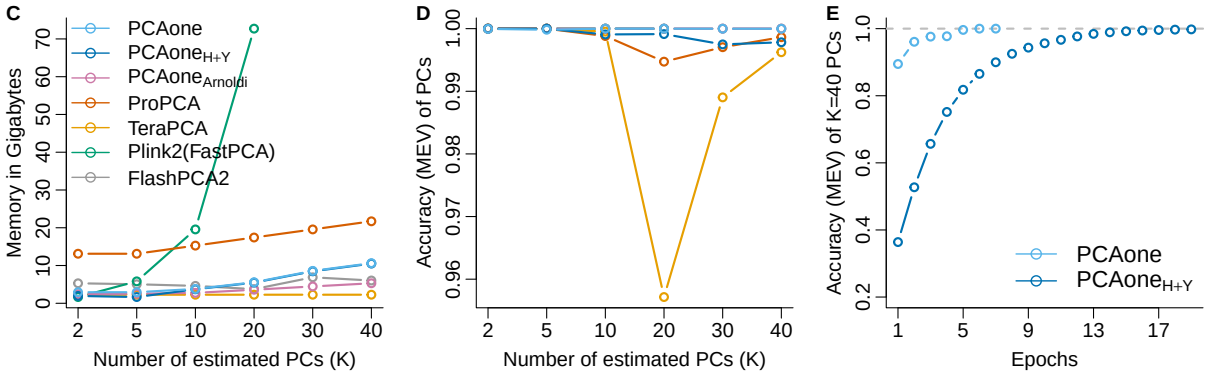

Figure S5: Performance on 1000 Genomes Project dataset with all populations. PCA performance by different software with varying number of inferred PCs (K) based on 5,766,022 SNPs and 2504 individuals. **A**, Number of epochs used by each software. *Not out-of-core, only allows for in memory computation with this number of iterations. **C**, Memory usage as a function of K. **D**, Accuracy (MEV) compared to the true full-rank SVD as a function of K. **E**, Convergence of PCAone and PCAone$_{H+Y}$ shown as accuracy per epoch.

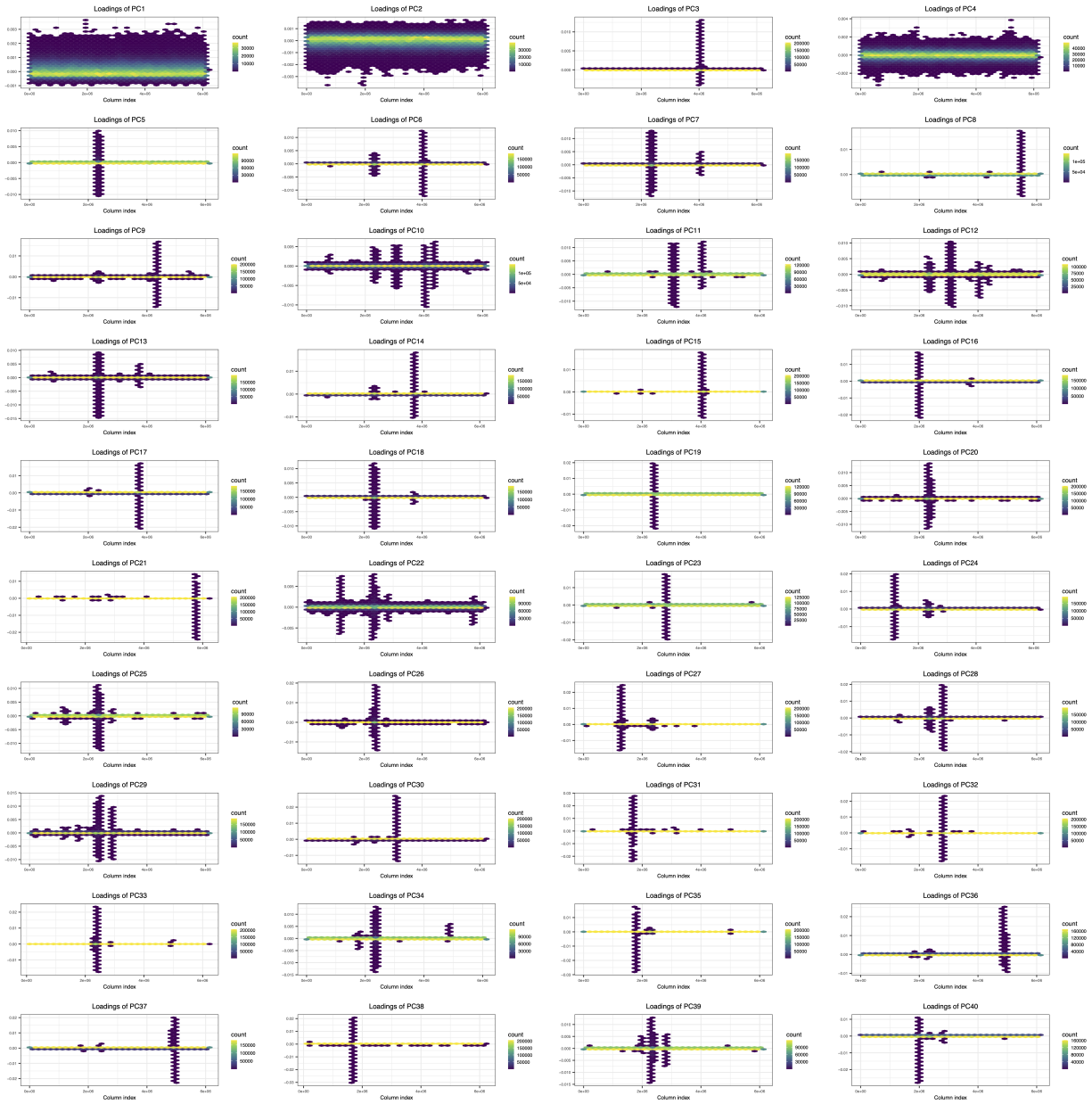

Figure S6: Top 40 PC loadings of the UK Biobank data with 487,409 individuals and 6,133,304 SNPs. Variants are ordered by chromosome and physical position, represented on the x-axis by their SNP number (Column index), and the value of loadings is represented on the y-axis. Points are hex-binned and colored by their density.

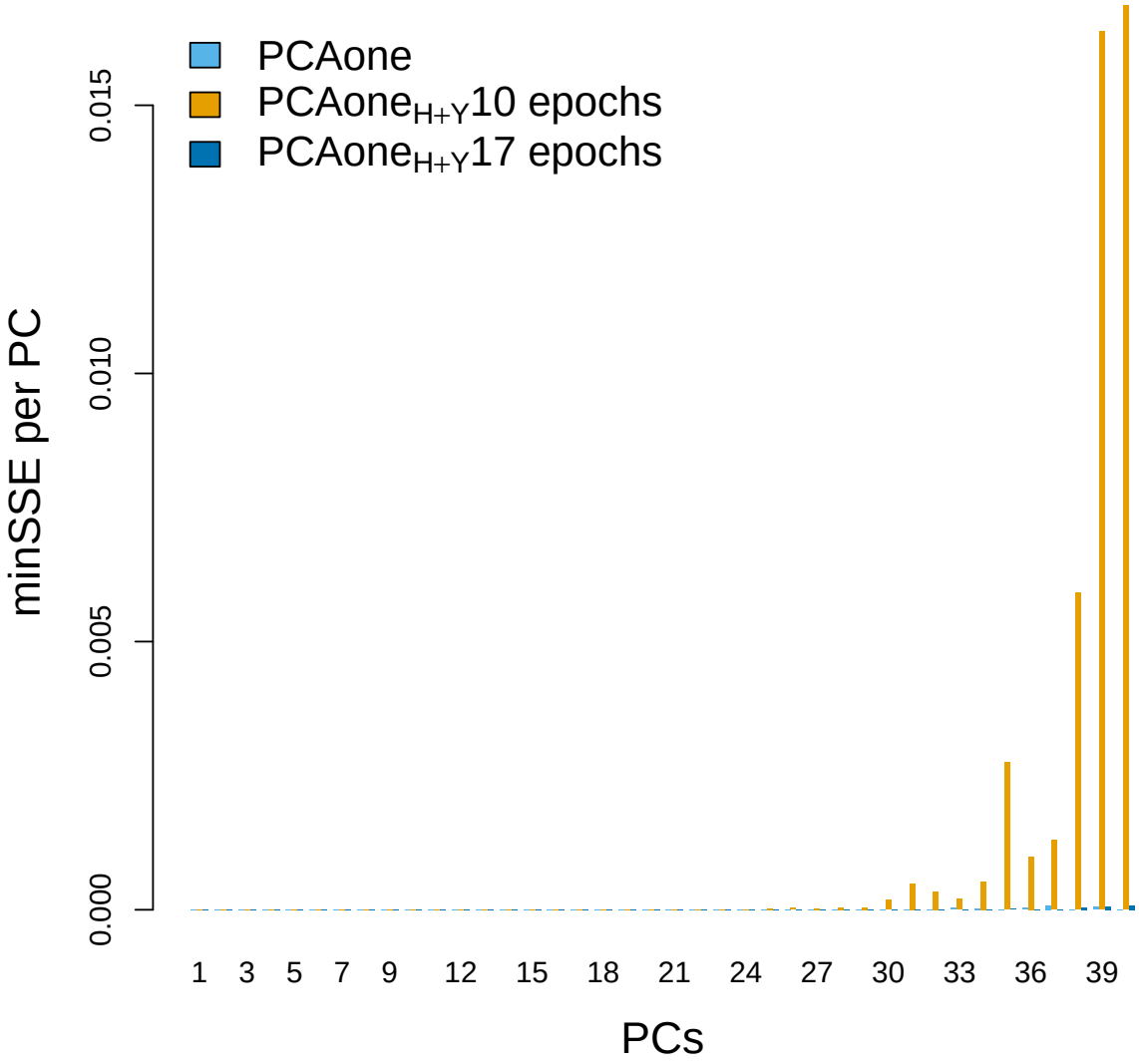

Figure S7: minSSE of top 40 PCs of the UK Biobank data with 487,409 individuals and 6,133,304 SNPs. minSSE were calculated based on each PCs the lowest distance to the PCs from PCAone$_{Arnoldi}$

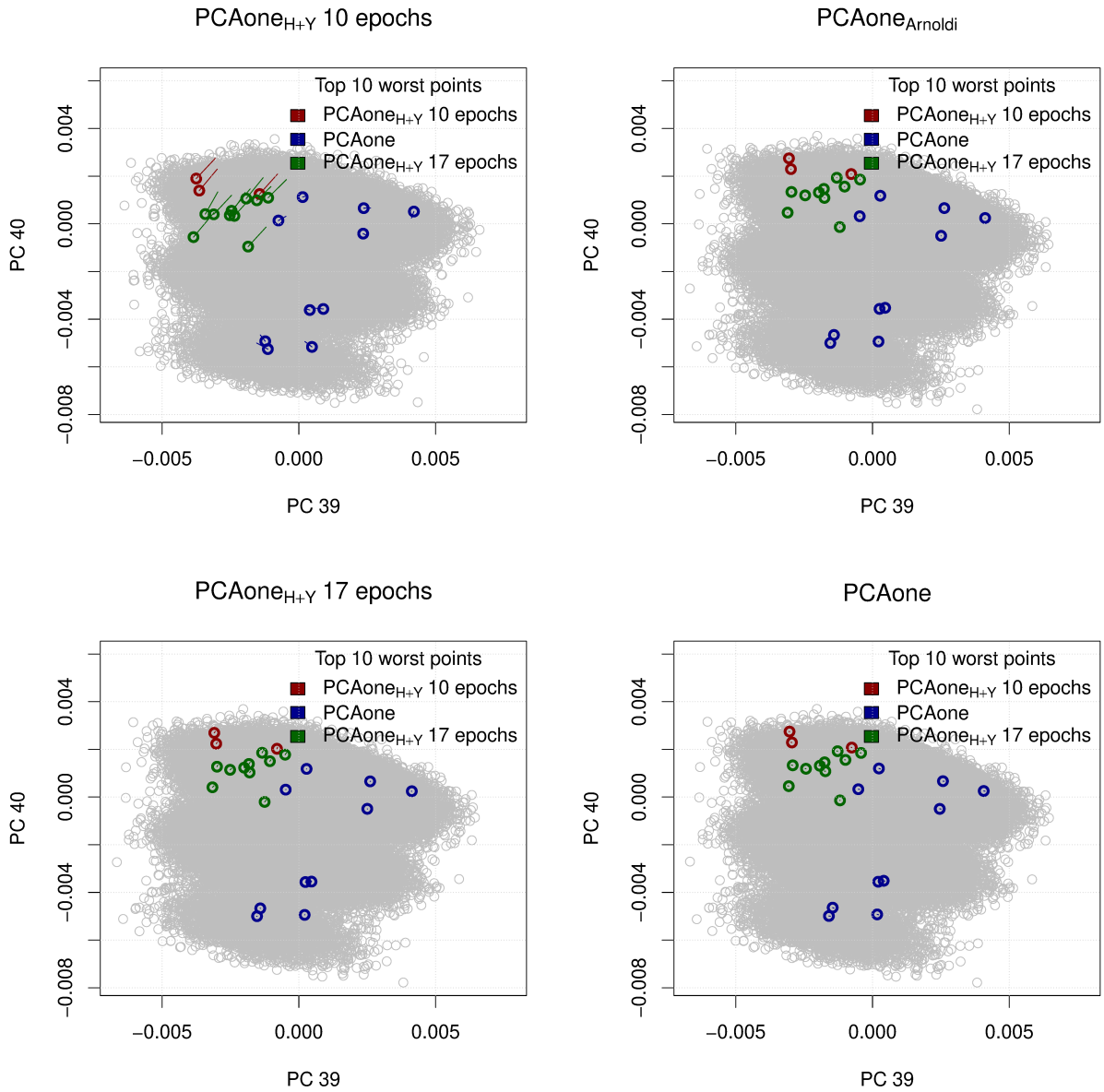

Figure S8: PC 39 and 40 of the UK Biobank data with 487,409 individuals and 6,133,304 SNPs. The top 10 points with the worst distance (minSSE) to the results from PCAone$_{Arnoldi}$ are highlighted and with a line that starts in the position in the PCAone$_{Arnoldi}$ coordinates.

S2 Supplementary tables

Table S1: MEV of top 40 PCs for 13 random subsets of UK Biobank imputed data in Figure 2. For large data sets it is not computationally feasible to perform the full-rank SVD and we therefore use PCAone$_{Arnoldi}$ as the truth when calculating the accuracy which is indicated by "-". Some analysis did not finish due to being out of memory (oom) and out of time(oot).

N	M	PCAone	PCAone$_{H+Y}$	Plink2	TeraPCA	ProPCA	FlashPCA2	PCAone$_{Arnoldi}$ *
10K	3M	1.0000	0.9984	oom	0.9978	0.9997	1.0000	-
20K	3M	1.0000	0.9993	oom	0.9980	1.0000	1.0000	-
40K	3M	1.0000	0.9995	oom	0.9995	1.0000	1.0000	-
60K	3M	1.0000	0.9996	oom	0.9995	1.0000	1.0000	-
80K	3M	1.0000	0.9996	oom	0.9995	1.0000	oot	-
100K	3M	1.0000	0.9998	oom	0.9996	1.0000	oot	-
50K	200K	0.9997	0.9994	1.0000	0.9994	1.0000	1.0000	1.0000
50K	400K	1.0000	0.9996	1.0000	0.9989	1.0000	1.0000	1.0000
50K	600K	1.0000	0.9997	1.0000	0.9995	1.0000	1.0000	1.0000
50K	800K	1.0000	0.9993	1.0000	0.9993	0.9999	1.0000	1.0000
50K	1M	1.0000	0.9996	1.0000	0.9996	1.0000	1.0000	1.0000
50K	2M	1.0000	0.9995	oom	0.9992	1.0000	1.0000	1.0000
50K	3M	1.0000	0.9995	oom	0.9995	1.0000	1.0000	1.0000

Table S2: minSSE of top 40 PCs for 13 random subsets of UK Biobank imputed data in Figure 2. * For large data sets it is not computationally feasible to perform the full-rank SVD and we therefore use PCAone$_{Arnoldi}$ as the truth when calculating the accuracy which is indicated by "-". Some analysis did not finish due to being out of memory (oom) and out of time(oot).

N	M	PCAone	PCAone$_{H+Y}$	Plink2	TeraPCA	ProPCA	FlashPCA2	PCAone$_{Arnoldi}$ *
10K	3M	0.0014	0.2695	oom	0.2086	0.0266	1e-10	-
20K	3M	0.0016	0.0630	oom	0.1875	0.0020	1e-10	-
40K	3M	0.0014	0.0706	oom	0.0491	1e-4	1e-10	-
60K	3M	0.0021	0.1541	oom	0.0817	3e-4	1e-10	-
80K	3M	0.0014	0.1198	oom	0.0418	6e-5	oot	-
100K	3M	0.0012	0.0558	oom	0.0412	4e-5	oot	-
50K	200K	0.0279	0.0595	0.0003	0.1091	0.0018	1e-10	1e-10
50K	400K	0.0042	0.0422	0.0002	0.1779	0.0010	1e-10	1e-10
50K	600K	0.0033	0.0348	0.0001	0.0498	1e-5	1e-10	1e-10
50K	800K	0.0051	0.0954	0.0002	0.2134	0.0070	1e-10	1e-10
50K	1M	0.0033	0.0449	0.0001	0.0746	1e-4	1e-10	1e-10
50K	2M	0.0025	0.0574	oom	0.1021	6e-4	1e-10	1e-10
50K	3M	0.0013	0.0769	oom	0.0566	3e-4	1e-10	1e-10

Table S3: MEV against PCs estimated by full-rank SVD (bold) and PCAone$_{Arnoldi}$ as a function of top PCs for scRNAs data with 12000 cells and 23771 genes. The default three iterations was used for OnlinePCA$_{Halko}$.

PCs	PCAone		PCAone$_{H+Y}$		OnlinePCA$_{Halko}$	
2	**1.0000000**	1.0000000	**1.0000000**	1.0000000	**0.9753135**	0.9753135
3	**1.0000000**	1.0000000	**1.0000000**	1.0000000	**0.9804843**	0.9804843
4	**1.0000000**	1.0000000	**1.0000000**	1.0000000	**0.9824888**	0.9824888
5	**1.0000000**	1.0000000	**1.0000000**	1.0000000	**0.9859905**	0.9859905
6	**1.0000000**	1.0000000	**1.0000000**	1.0000000	**0.9879956**	0.9879956
7	**1.0000000**	1.0000000	**1.0000000**	1.0000000	**0.9897004**	0.9897004
8	**1.0000000**	1.0000000	**1.0000000**	1.0000000	**0.9848932**	0.9848932
9	**1.0000000**	1.0000000	**1.0000000**	1.0000000	**0.9889352**	0.9889352
10	**1.0000000**	1.0000000	**1.0000000**	1.0000000	**0.9900309**	0.9900309
11	**1.0000000**	1.0000000	**1.0000000**	1.0000000	**0.9903505**	0.9903505
12	**1.0000000**	1.0000000	**1.0000000**	1.0000000	**0.9912341**	0.9912341
13	**1.0000000**	1.0000000	**1.0000000**	1.0000000	**0.9918151**	0.9918151
14	**1.0000000**	1.0000000	**1.0000000**	1.0000000	**0.9891819**	0.9891819
15	**1.0000000**	1.0000000	**0.9999999**	0.9999999	**0.9703988**	0.9703988
16	**1.0000000**	1.0000000	**0.9999998**	0.9999998	**0.9913479**	0.9913479
17	**0.9999999**	0.9999999	**0.9999996**	0.9999996	**0.9899419**	0.9899419
18	**0.9999999**	0.9999999	**0.9999987**	0.9999987	**0.9890437**	0.9890437
19	**0.9999997**	0.9999997	**0.9999976**	0.9999976	**0.9698647**	0.9698647
20	**0.9999995**	0.9999995	**0.9999940**	0.9999940	**0.9817112**	0.9817112
Truth	full SVD	PCAone$_{Arnoldi}$	full SVD	PCAone$_{Arnoldi}$	full SVD	PCAone$_{Arnoldi}$

Table S4: Stopping criterion used by different programs. For each epoch i, $V^{(i)}$ denotes matrix of sigularvectors, $s^{(i)}$ denotes vector of singularvalues, $p^{(i)}$ denotes likelihood. k denotes number of top PCs to be estimated by the program.

Program	Stopping criterion
PCAone	$1 - mev(V^{(i)}, V^{(i-1)}) < 1e-4$
PCAone$_{H+Y}$	$1 - mev(V^{(i)}, V^{(i-1)}) < 1e-4$
FastPCA/Plink2	k number of iterations
TeraPCA	$abs((s^{(i)} - s^{(i-1)})/s^{(i)}) < 1e-3$
ProPCA	$p^{(i)} - p^{(i-1)} < 1e-3$

Table S5: Algorithm's Complexity. n denotes the number of samples. m denotes the number of SNPs. b denotes the block size. k denotes the number of top PCs to be estimated. $l = k + 10$.

Program	Memory complexity	Time complexity per epoch
PCAone$_{Arnoldi}$/FlashPCA2	$O(nb)$	$O(nk^2)$
PCAone	$O(mb + 2ml + 3nl)$	$O(2nml)$
PCAone$_{H+Y}$	$O(mb + 2ml + 3nl)$	$O(nml)$
TeraPCA	$O(mb + 2ml)$	$O(nml)$
FastPCA/Plink2	$O(nm + m(k+1)l)$	$O(nml)$
ProPCA	$O(nm)$	$O(nml/log_3(max(n,m))$