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

June 6, 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. Here we present PCAone, a PCA framework with three PCA algorithms and an unified interface for different input data. We propose a new RSVD algorithm that can achieve high accuracy without passing through the data many times. Standard RSVD algorithms use the whole data matrix in each optimization step which makes out-of-core implementations slow. In contrast, PCAone uses a window based optimization scheme based on blocks of data which allows the algorithm to converge within a few passes through the whole 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 13.5 GBs memory and 20 CPU threads to calculate the top 40 PCs 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 35 minutes to capture the top 20 PCs accurately. The software 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 (N Patterson et al. 2006) and clustering cells from single-cell sequencing experiments (Kiselev et al. 2019). The estimated principal components (PCs) are also often 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, NJ 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 and Hinton 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 popular 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 and makes it computationally less efficient. Therefore all of the fast algorithms read through the data many times. 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 PCAone, a fast and accurate out-of-core framework written in C++ introducing a new algorithm which improves both the accuracy and the speed of the fast RSVD approach as well as including efficient PCA implementations of existing algorithms. The framework is designed to efficiently process large-scale and high-dimensional data of different formats for biological researches. As RSVD is more pass-efficient but less accurate than IRAM, we propose 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. We have implemented the new RSVD algorithm in our software PCAone. In order to compare our algorithm to existing algorithms, we also made efficient implementations of the IRAM and RSVD algorithm in PCAone as well. The methods are implemented both as 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 only focus on the out-of-core implementation, which is the only option for the large-scale datasets. Our implementations of the existing IRAM and RSVD methods are made efficient by taking advantage of full parallelization and the high performance numerical library MKL (`https://www.intel.com/content/www/us/en/developer/tools/oneapi/onemkl.html`) written in C++.

2 Results

2.1 Overview of PCAone framework

We implemented three out-of-core algorithms in our software PCAone. An IRAM algorithm, PCAoneA, inspired by the blockwise approach in FlashPCA2 (Abraham et al. 2017) but with multithreading capabilities. Likewise, we implemented the much faster RSVD algorithm here denoted as PCAoneH, which uses 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. Our new proposed RSVD algorithm, called PCAoneF, applies a window based power iteration scheme on permuted subsets of the input matrix, such that power iterations are performed multiple times for each pass over the data as illustrated in Figure S1 (Step 1 PCAone). This is in contrast to the normal RSVD algorithm where the whole input matrix is read twice during each power iteration (Figure S1 Step1 Halko). See the details of our PCAone algorithms and architectures in Methods and supplementary Figure S2.

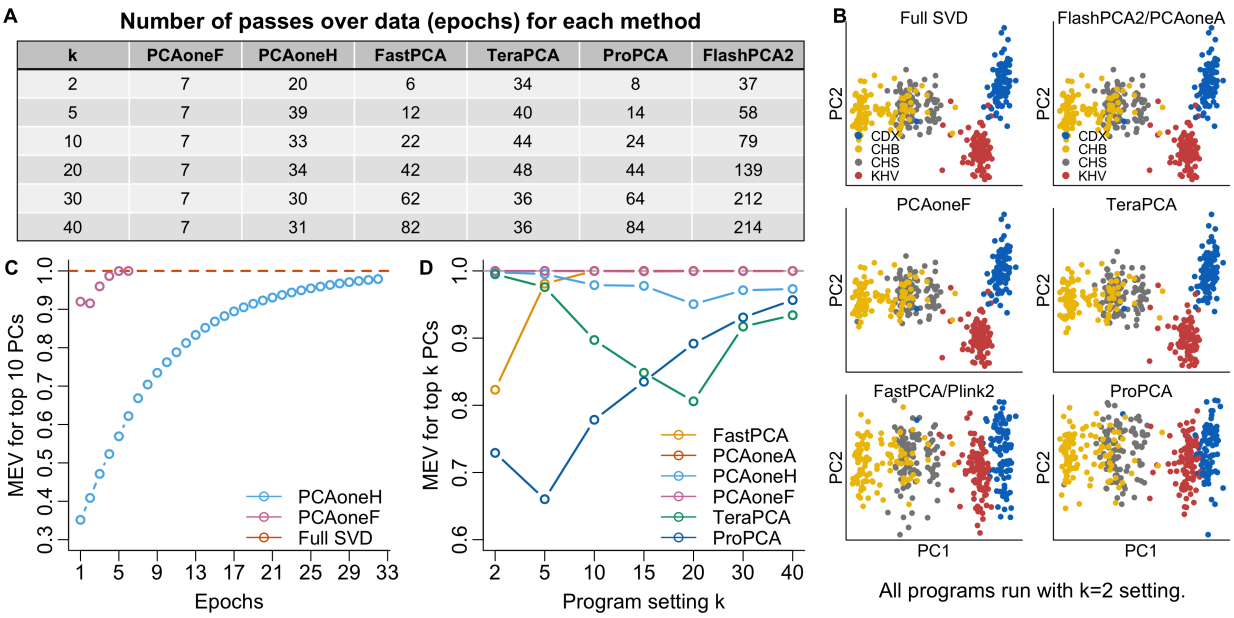
2.2 Accuracy and iterations

We first evaluated the performance on genetic datasets that are small enough for us to perform PCA using full-rank SVD to compare the different methods. The performance of the different methods will depend on the number of observations e.g. individuals, number of features e.g. SNPs, as well as the amount of variance explained by the top PCs. 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. We compared our three implemented algorithms (PCAoneA, PCAoneH, PCAoneF) with software for performing large-scale PCA in genetics, FlashPCA2, FastPCA, Plink2 (faster reimplementation of FastPCA, Chang et al. 2015), TeraPCA and ProPCA. To measure the accuracy, we compute the mean explained variance (MEV) between the estimated PCs and that of the full-rank SVD.

As the Asian populations are genetically similar and the top eigenvalues of the data are very small and decay slowly (Figure S3), we expect the accuracy of the PC approximation to be important and hard to achieve (Halko et al. 2011; Yu et al. 2017). The number times the data is read through (epochs) to obtain the top k PCs is shown in Figure 1A. As can be seen, our proposed PCAoneF algorithm consistently uses only 7 epochs for all k settings while the other algorithm uses more and more epochs for higher values of k, which is related to both the algorithms' efficiency and convergence criteria. Most of the methods produce almost identical top 2 PCs compared to the full-rank SVD in a $k = 2$ setting but FastPCA/Plink2 and ProPCA failed to accurately capture the difference between the Dai and Vietnamese (Figure 1B). The main reason for the low accuracy of FastPCA/Plink2 is their stopping criteria which is only based on the choice of k, where the accuracy increases for higher choices of k. Our RSVD implementations, PCAoneH and PCAoneF, use a stopping criterion based on the difference in inferred PCs between successive epochs (see Methods and Table S7). Using the same stopping criterion to estimate top 10 PCs, we showed that PCAoneF can achieve high accuracy (MEV > 0.9999) using only 6 epochs while the normal Halko (implemented in PCAoneH) used 32 epochs (Figure 1C). This demonstrates that PCAoneF can reduce the number passes over data significantly while achieving high accuracy. We compared the accuracy across different choices of k (Figure 1D) which showed that PCAoneF and the IRAM methods (PCAoneA, FlashPCA2) consistently had high accuracy. Regarding the other RSVD methods, PCAoneH always performed better than TeraPCA and FastPCA/Plink2 only achieved high accuracy with high choices of k which is explained by their difference in stopping criteria and therefore, their number of epochs. And the probabilistic based method ProPCA did not achieve as high accuracy at any k on this dataset.

Additionally, we analyzed the accuracy based on all populations from 1000 Genomes Project. In this dataset there is much more pronounced population structure and much more of the variation explained by the top PCs. Here all methods were able to estimate PCs with higher accuracy (MEV>0.95) across a range of chosen number of PCs (see Table S1). Likewise, all methods had high accuracy when applied to subsets of the of UK Biobank genetic data regardless of the number of samples or the number of imputed variants (see Table S2 and Table S3), but PCAoneF always uses only 5-7 epochs compared to more than 40 iterations for TeraPCA, FastPCA/Plink2, and ProPCA. As the full-rank SVD is not computationally feasible to run on these bigger datasets and the IRAM method are very accurate for capturing top PCs (Privé, Aschard, et al. 2018; Tsuyuzaki

et al. 2020), we use PCs estimated by PCAoneA as the standard when evaluating the accuracy of the other methods for large datasets.



k	PCAoneF	PCAoneH	FastPCA	TeraPCA	ProPCA	FlashPCA2
2	7	20	6	34	8	37
5	7	39	12	40	14	58
10	7	33	22	44	24	79
20	7	34	42	48	44	139
30	7	30	62	36	64	212
40	7	31	82	36	84	214

Figure 1: Accuracy and performance. **A**, compared the number of epochs used by different methods given a range of k setting and default stopping criteria. Note PCAoneA and Plink2 are faster reimplementation of FlashPCA2 and FastPCA, respectively. **B**, showed the accuracy of top 2 PCs for each method given $k = 2$ setting compared to full-rank SVD. **C**, compared the performance of PCAoneF and PCAoneH capturing top 10 PCs as a function of epochs used. **D**, showed the accuracy of all methods as a function of program setting k.

2.3 Runtime and memory usage

We assessed the scalability and performance of PCAone with various large real-world datasets, which are randomly subseted from the UK Biobank genetic data. We compared the wall-clock time and memory usage to calculate the top k=20 PCs. As shown in Figure 2 , all methods had a mostly linear relationship between number of individuals or SNPs compared to wall-clock run time while the memory usage was near constant for the out-of-core implementations (PCAone, FlashPCA2, TeraPCA) and linear for ProPCA and Plink2. Our multithreaded IRAM implementation, PCAoneA, is up to 10 times faster than FlashPCA2 using 20 threads. Similarly our out-of-core RSVD implementations, PCAoneH and PCAoneF, are much faster than FastPCA/PLINK2 despite using much less memory by reading data stored on the disk. The fasted method, PCAoneF, is on average 2x, 5x, 9x, 12x, 30x faster than PCAoneH, Plink2, ProPCA, TeraPCA, FlashPCA2, respectively, using only a few GBs of memory.

Additionally, we further investigated the performance of all software on calculating the top 40 PCs for our biggest dataset with 487,409 individuals and 6,133,304 SNPs from the UK Biobank. As ProPCA and Plink2 consume more memory than 192 GB, we can only run PCAone, TeraPCA and FlashPCA2 on this dataset due to the limited memory setting. We summarized all cost in details in Table 1. First of all, only PCAoneF and PCAoneH can be finished within a day with a runtime of 9.0 hours and 14.2 hours, respectively. We note that PCAoneF requires the data to be permuted first in order to work efficiently, which took one hour. If the data was already permuted, it would only had taken 8.0 hours. It took 49.3 hours for TeraPCA to finish and 4 days for PCAoneA while FlashPCA2 was not able to finish in one week. We also calculated MEV for the three RSVD implementations using PCs estimated by PCAoneA as the standard which showed that PCAoneF was not just the fastest of the RSVD method but also the most also the accurate.

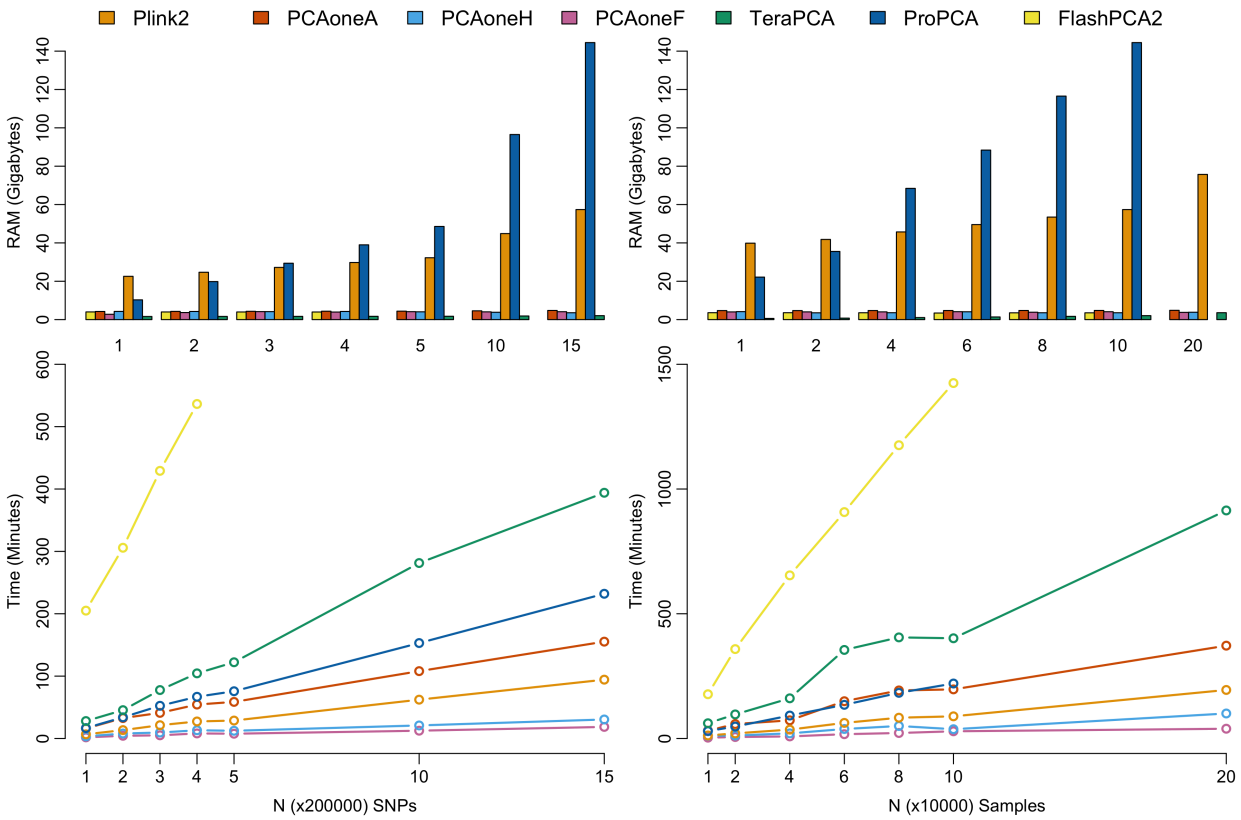


Figure 2: Benchmarking of runtime and memory usage. The left column shows the runtime and memory usage of calculating top k=20 PCs for the datasets whose number of samples is 100,000 and number of SNPs varies from 200,000 to 3,000,000, while the right column fix number of SNPs as 3,000,000 and scales number of samples from 10,000 to 200,000. For a fare comparison, we use the parameters given best performance on both speed and memory for each program as possible as we can. The number of threads is 20 for all except FlashPCA2 and most other parameters remain as default for each program.

Table 1: Runtime of capturing top 40 PCs of UK Biobank data with 487,409 individuals and 6,133,304 SNPs. Time is in hour unit. RAM is in gigabyte unit. * refer to IO time for permuting data. Here PCs estimated by PCAoneA is used as the standard for calculating MEV. All programs used 20 threads.

Program	Wall Time(h)	IO Time(h)	Iterations	RAM(GB)	MEV(k=40)
PCAoneF	9.0	5.8+1*	5	13.55	0.9999969
PCAoneH	14.2	7.6	10	13.21	0.9996642
TeraPCA	49.3	10.5	9	9.4	0.9994317
PCAoneA	105.3	87.1	124	7.05	-

2.4 Analyses on UK Biobank data

To show the application of PCAone on biobank scale genetic data, 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. Using 40 threads and 13.55 GBs memory, it took 9 hours for PCAoneF to calculate the top 40 PCs. And we investigated the distribution of the PCs loadings. As shown in Figure S4 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 known skin pigmentation gene *SLC45A2* (Beleza et al. 2013). 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.

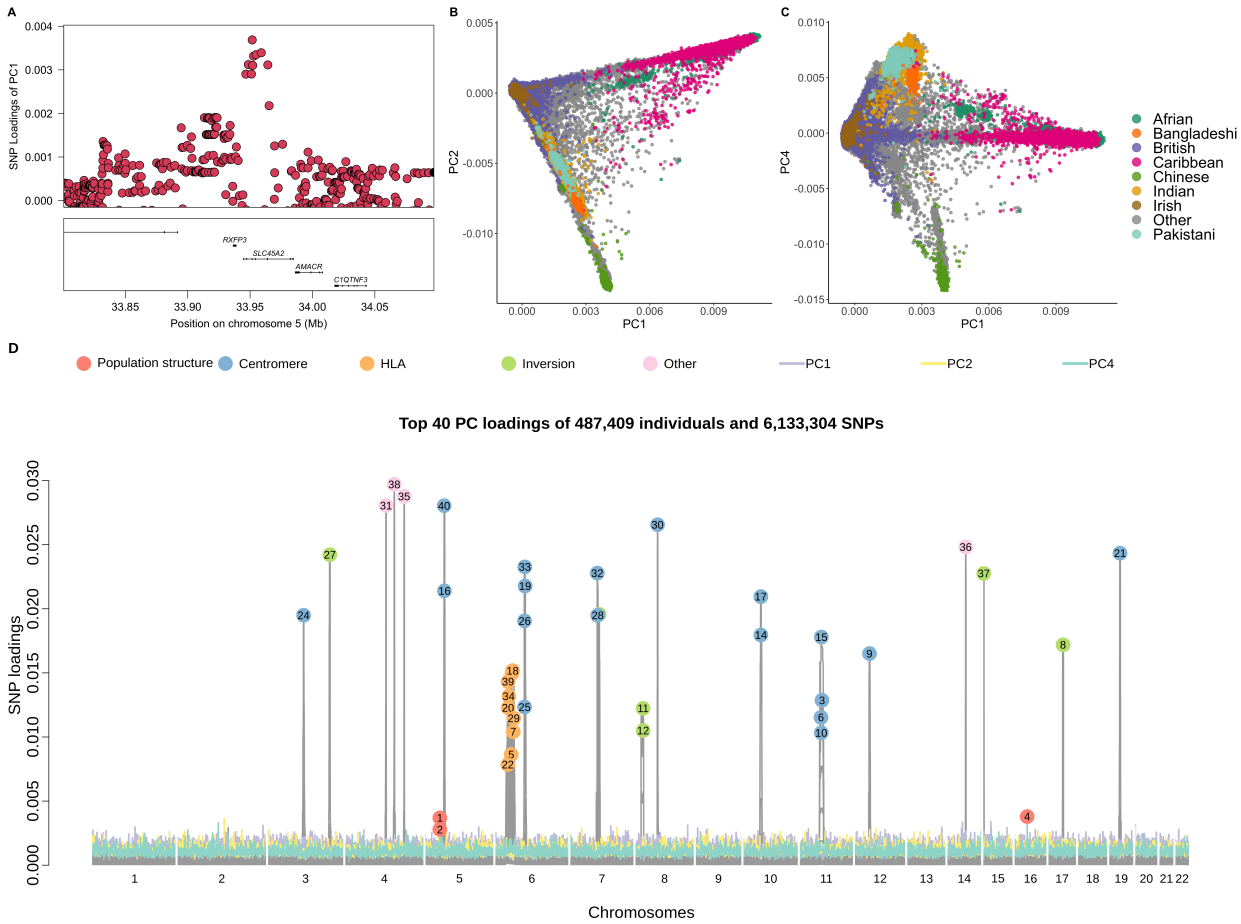


Figure 3: PCA of the whole UK biobank imputed dataset. **A**, showed the locuszoom of variant rs16891982 with highest loadings for PC1 and PC2. **B** and **C** showed the population structure of PC1 vs PC2 and PC1 vs PC4 respectively for all 487,409 individuals, colored by country or region label. other labels are merged into the category "other". **D**, summarized the distribution of SNP loadings for top 40 PCs. Peeks are annotated and colored by the region they located in.

2.5 Analyses on single cell data

In PCAone we have not only implemented multiple PCA algorithms but we also support 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 scRNAs datasets with 1,306,127 mice brain cells and 23,771 genes, which took 35 minutes for PCAoneF to estimate the top 20 PCs using 4 GBs of memory. We compared the performance of PCAone with the OnlinePCA.jl-halko method which a recent study showed was the fastest out-of-core implementation available for scRNAs data (Tsuyuzaki et al. 2020). As different software uses different input formats, such as the CSV format (oocRPCA, R) and binarized matrix further compressed by Zstd (OnlinePCA.jl, Julia), PCAone supports the general CSV format compressed by Zstd as input assuming the data is normalized so that users can apply their own normalization method for their special data. Moreover, PCAone 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. Therefore, we evaluated the time for I/O operations, normalization and PCA separately for each program. As shown in Figure 4A, PCAone used less time for I/O operations and preprocessing than OnlinePCA.jl. The benchmarking showed that PCAone is 4x-8x times faster than OnlinePCA.jl although both used few GBs of memory.

As full SVD is not computationally feasible and IRAM is considered the most accurate one for estimating top PCs, we therefore used the results of PCAoneA as standard when calculating MEV (see Figure 4B) which shows that PCAoneF and PCAoneH has higher accuracy. 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. Here the accuracy is the same when we substitute the full SVD for PCAoneA (Table S5 and Table S6). Lastly, we performed UMAP analyses with the top 10 PCs estimated by PCAoneA as input (see Figure 4). The UMAP results changed with difference seeds so we did not try to compare the UMAP projections from different PCA methods. Additionally, PCAoneA is the only out-of-core implementation of IRAM that can be performed on large scale scRNAs datasets.

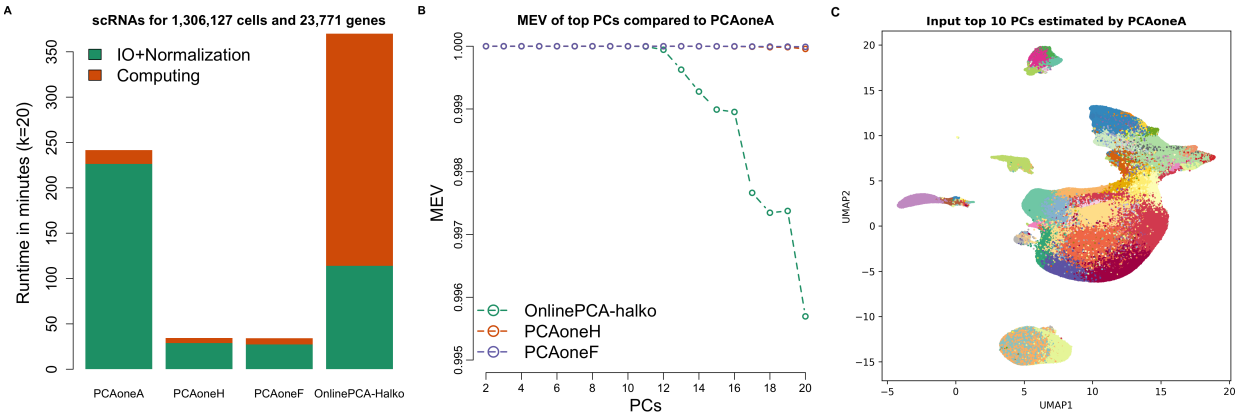


Figure 4: Performance on single cell RNAs data with raw counts for 1,306,127 cells and 23,771 genes. Top k=20 PCs were inferred and the default setting of 3 iterations was applied for OnlinePCA-halko. **A**, Runtime performance split into time used for I/O and for computation. **B**, Accuracy measured as MEV of various top PCs for each RSVD method compared to the IRAM method PCAoneA. **C**, UMAP plot based on top 10 PCs estimated by PCAoneA. Each cell is colored according to their cluster membership.

3 Discussion

We have presented PCAone, an out-of-core PCA framework designed for large-scale data analysis. Based on multiple data set we have demonstrated that our IRAM implementation, PCAoneA, is extremely accurate and much faster than the other accurate IRAM method FlashPCA2. In all analysis our new RSVD algorithm, PCAoneF, has higher accuracy as existing RSVD algorithms while being faster. When the top PCs explain very little of the variance in the data, PCAoneF needs far fewer number of epochs to converge, thus making it much faster than other 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 was much higher for the other methods while PCAoneF only needed 7. For this dataset the accuracy of different methods compared against the full-rank SVD varied a lot. Unsurprisingly the IRAM methods, PCAoneA and FlashPCA2, performed the best but the much faster PCAoneF algorithm performed almost as well and produced PCs that were indistinguishable from the full SVD. The other algorithms based on RSVD performed much worse for the East Asian dataset. The performance of FastPCA/Plink2 and ProPCA increased when many PCs were estimated which is not surprising since their number of inner iterations is increasing as well. Most of the methods can be run with user defined stopping criteria which improve their general performance at the expense of speed. PCAoneF often uses exactly 7 epochs despite also 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 iterations that use all of the data.

We benchmarked the performance of PCAone in terms of speed and memory based on different subsets of the imputed genotype data of the UK Biobank. PCAoneF is the fastest algorithm followed by the other RSVD methods, PCAoneH and FastPCA/Plink2. FastPCA/Plink2 and ProPCA are not out-of-core implementations and therefore 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 standard RSVD implementation PCAoneH is faster than FastPCA/Plink2 despite having I/O operations being more time consuming than computations. Part of the explanation is that the number of iterations used by FastPCA/Plink2 is around 4 and 8 times more than for PCAoneH and PCAoneF, respectively.

To illustrate how PCAone can be useful, we analyzed all of the 6 million common imputed variants in the UK Biobank consisting of almost half a million individuals. This took less than 10 hours using 13.5 GBs of memory and 20 CPU threads. 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 a single cell RNA sequencing dataset with 1,306,127 mice brain cells and 23,771 genes, which only took 35 minutes for PCAoneF to calculate the top 20 PCs using 4 GBs memory. PCAoneF was both much faster and more accurate than the recently proposed OnlinePCA.jl-halko method which was shown to outperform other methods used in scRNAs data (Tsuyuzaki et al. 2020).

Beyond the 3 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, genotype dosages and genotype likelihood data, 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. And the PCAone framework is highly optimized for out-of-core computations and implemented as a stand alone C++ software.

4 Methods

4.1 PCAone algorithms

As our proposed algorithm is in the spirit of Halko et al. 2011 randomized SVD, we therefore briefly explain the underlying idea, 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 left eigenvectors of A are approximated by the left eigenvectors of the low dimensional Q^TA. A basis Q can be found 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 orthogonalization $Q = qr(A\Omega)$ on this product. To find a Q that can yield a better approximation, Ω can be optimized by another orthogonalization process $\Omega = qr(A^TQ)$. Those two computations compose the so-called power iteration step where each power iteration goes through the whole matrix A twice. Recently, Yu et al. 2017 proposed a single pass RSVD algorithm that can go through the input matrix only once instead of twice for each iteration. This

is archived by reading a block of data once and then performing two matrix multiplications on it. We simplify Yu's algorithm for our own use in Algorithm 1, which is our implementation for PCAoneH. Note that when we want to estimate k PCs we will use an near optimal bases Q that has $l = k + 10$ columns. This padding alleviates some of the issues of eigenvalues switching order in the optimization.

Algorithm 1 The one pass randomized SVD algorithm for PCAoneH simplified from Yu 2017.

Given: Input matrix $A \in \mathbb{R}^{m \times n}$, maximum p iterations, $l = k + 10$
Return: Top k eigenvectors of A

```
draw random matrix Ω^(0) ∈ R^(n×l) from a standard Guassian distribution
for i = 1 : p do
    G^(i) ← AΩ^(i−1); H^(i) ← A^T G^(i)        ▷ can be done in blockwise
    Ω^(i), R ← qr(H^(i))                        ▷ get optimal Ω by power iteration
end for
Q_1, R_1 ← qr(G^(p))
Q_2, R_2 ← qr(Q_1)
R ← R_1 R_2                                     ▷ re-orthogonalization step
B ← R^(−T) Ω̂^T                                  ▷ get low-dimentional B from optimal Ω̂
U, S, V ← svd(B)
return V_(1:k)
```

To reduce the computational burden of the power iterations, we propose a new power iteration scheme for large-scale datasets called PCAoneF, which reduces the amount of times the data is accessed. It is achieved by performing power iteration on subsets of the data as described in Algorithm 2. We first permute the input matrix by rows (features) and define $w = 64$ number of blocks with m/w number of row entries of A for each block. In this way each window of the data is a random subset of the features which is important for genetic data where SNPs are often locally correlated due to linkage disequilibrium. In the first epoch, we perform power iterations based on data in a sliding window of 2 blocks with a step size of one block. After reading through the whole data (one epoch), we double both the window size and step size until the window size equals the whole input matrix size. Therefore, power iteration is performed in the same way as the normal Halko's process after 6 epochs. The algorithm stops when the difference of eigenvectors between two successive epochs is smaller than an user defined threshold (default $1 - MEV < 1e - 4$). This often happens after 6 epochs where a second update of Ω based on the whole data is not needed. Each iteration we update Ω the sign of each column is random and we therefore match them between iterations. In practice, all the matrix multiplications can be performed blockwise so that we can only read a small piece of the data in memory.

Algorithm 2 The enhanced window based power iteration algorithm for PCAoneF

Given: Input matrix $A \in \mathbb{R}^{m \times n}$, maximum p iterations, $l = k + 10$
Return: Top k eigenvectors of A

1: permute the rows of A and divide A into $2^6 = 64$ blocks by rows
2: draw random matrix $\Omega^{(0)} \in \mathbb{R}^{n \times l}$ from a standard Guassian distribution
3: set $i = 1$, $H^{(0)} = H^{(-1)} = 0$
4: **for** $e = 1 : p$ **do**
5: set window size to $min(2^e, 64)$ blocks, define window index as set of rows w_i
6: **repeat**
7: $G^{(i)} \leftarrow A_{w_i}\Omega^{(i-1)}; H^{(i)} \leftarrow A_{w_i}^T G^{(i)}$
8: $H^{(i)} \leftarrow H^{(i)} + H^{(i-1)} - H^{(i-2)}$ ▷ **flip signs of** $H^{(i)}$ **columns to match** $H^{(i-1)}$
9: $\Omega^{(i)}, R \leftarrow qr(H^{(i)})$
10: $i = i + 1$
11: **if** $e < 6$ **then** ▷ A_{w_i} **is** A **when** $e \geq 6$
12: slide half a window size for next window w_i
13: **end if**
14: **until** step through the whole input data
15: **end for**
16: use 6–11 step in Algorithm 1 to obtain top k eigenvectors.

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). Assuming we have a true set of PCs $(v_1, v_2, ..., v_k)$, which we evaluate the accuracy of a set of estimated PCs $(u_1, u_2, ..., u_k)$ as following:

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

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

For some RSVD implementations, the users have to specify the number of power iterations themselves, which therefore lacks an explicit measure of accuracy. In 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 $1 - \text{MEV} \leq 1e - 4$ 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) with 487,411 individuals and 93,095,645 SNPs for the autosomes. 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 UMAP 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 378 GB of memory and 88 threads available (CPU model: Intel(R) Xeon(R) Gold 6152 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 denotes 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 denotes the normalized matrix where each entry $a_{i,j}$ is median-normalized and log-transformed as the following.

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

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

5 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.

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.

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

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

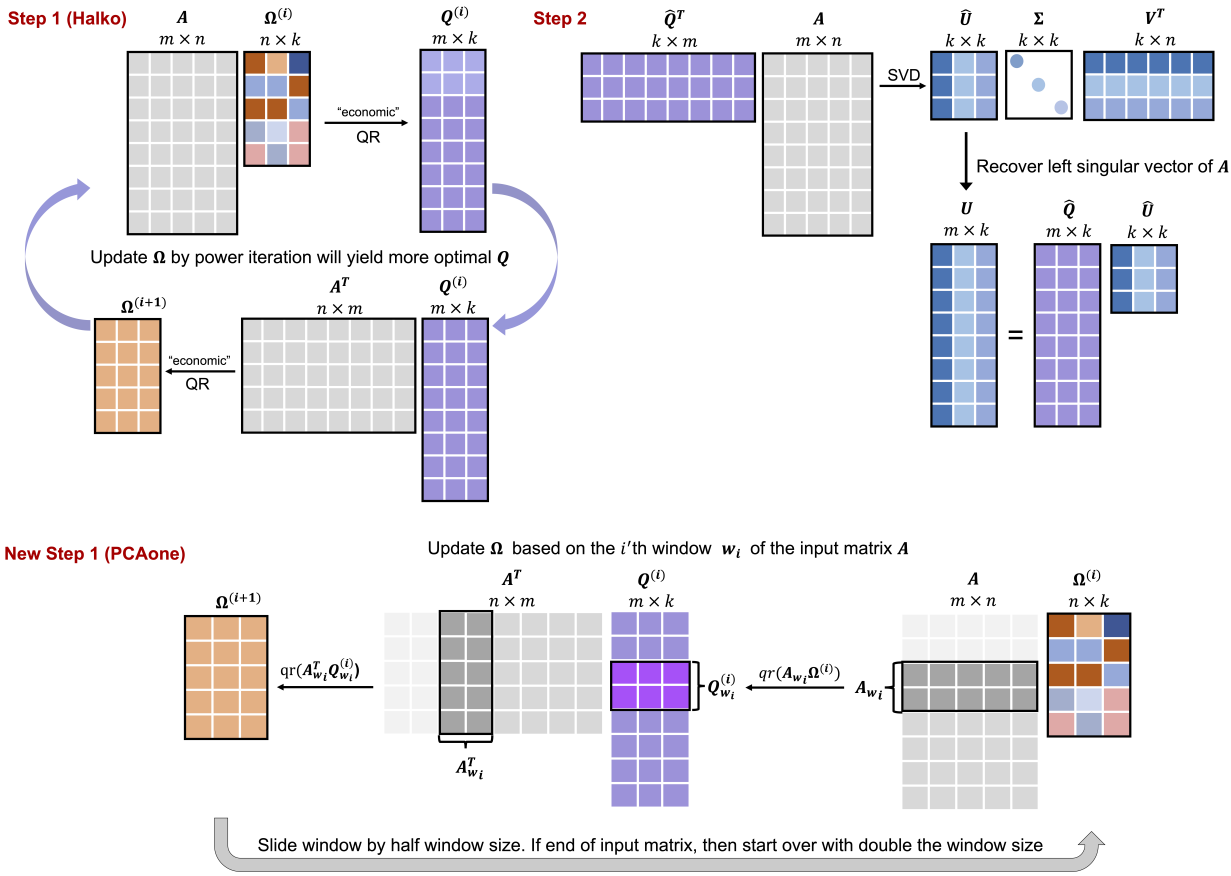


Figure S1: Overview of algorithms. The underlying idea of randomized SVD e.g. PCAoneH, is seeking a near-optimal orthonormal basis $Q^{m\times k}$ for input matrix $A^{m\times n}$, such that $A \approx QQ^TA$ is satisfied with $k \ll m, n$. In **step1 of the Halko**, Q is obtained from QR orthogonalization $Q = qr(A\Omega)$ 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.

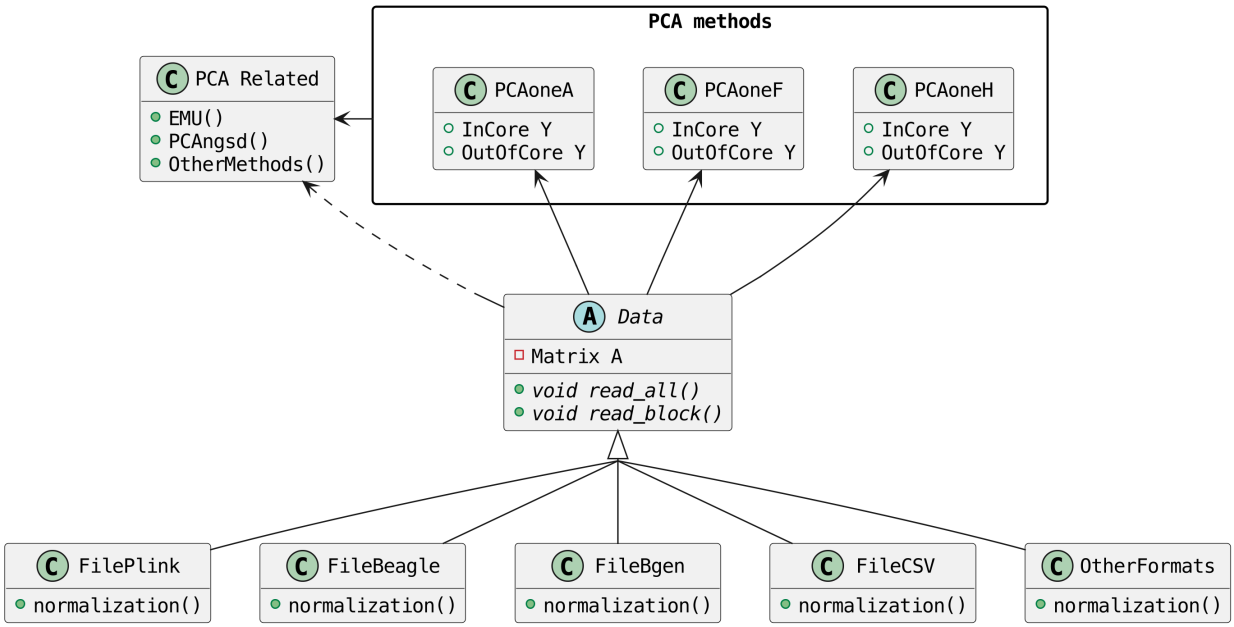


Figure S2: Overview of PCAone Architecture. Multiple file formats for diallelic genotype data (Plink binary), genotype likelihoods (beagle imputation format), allelic dosages from imputation (Bgen), count data such as scRNA (CSV) or a simple matrix of real values (CSV). PCAone also works with missing 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).

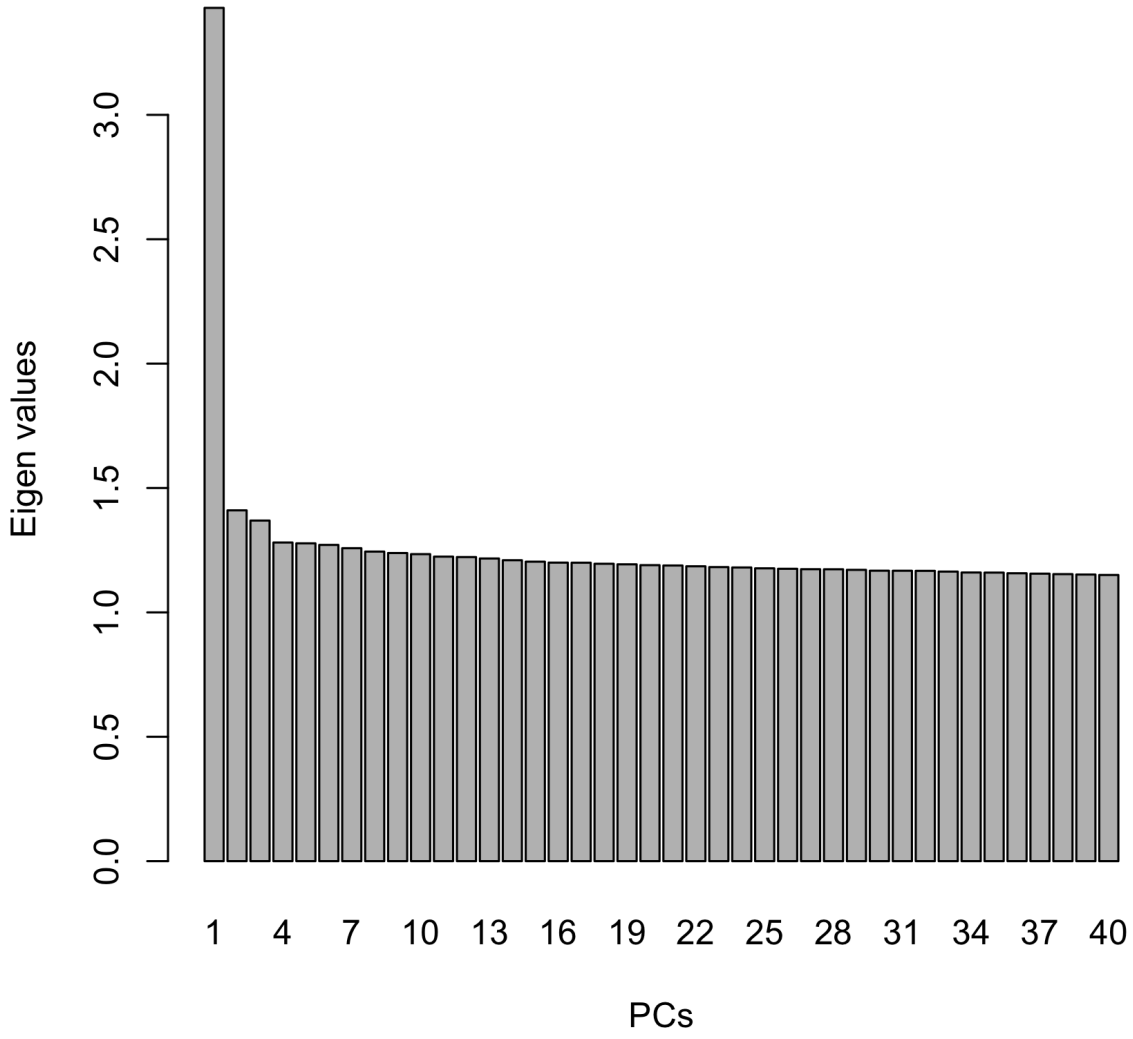


Figure S3: Top 40 eigenvalues from the full SVD of the 1000 genomes east Asian population CHB, CHS, CDX, KHV analyzed in Figure 1.

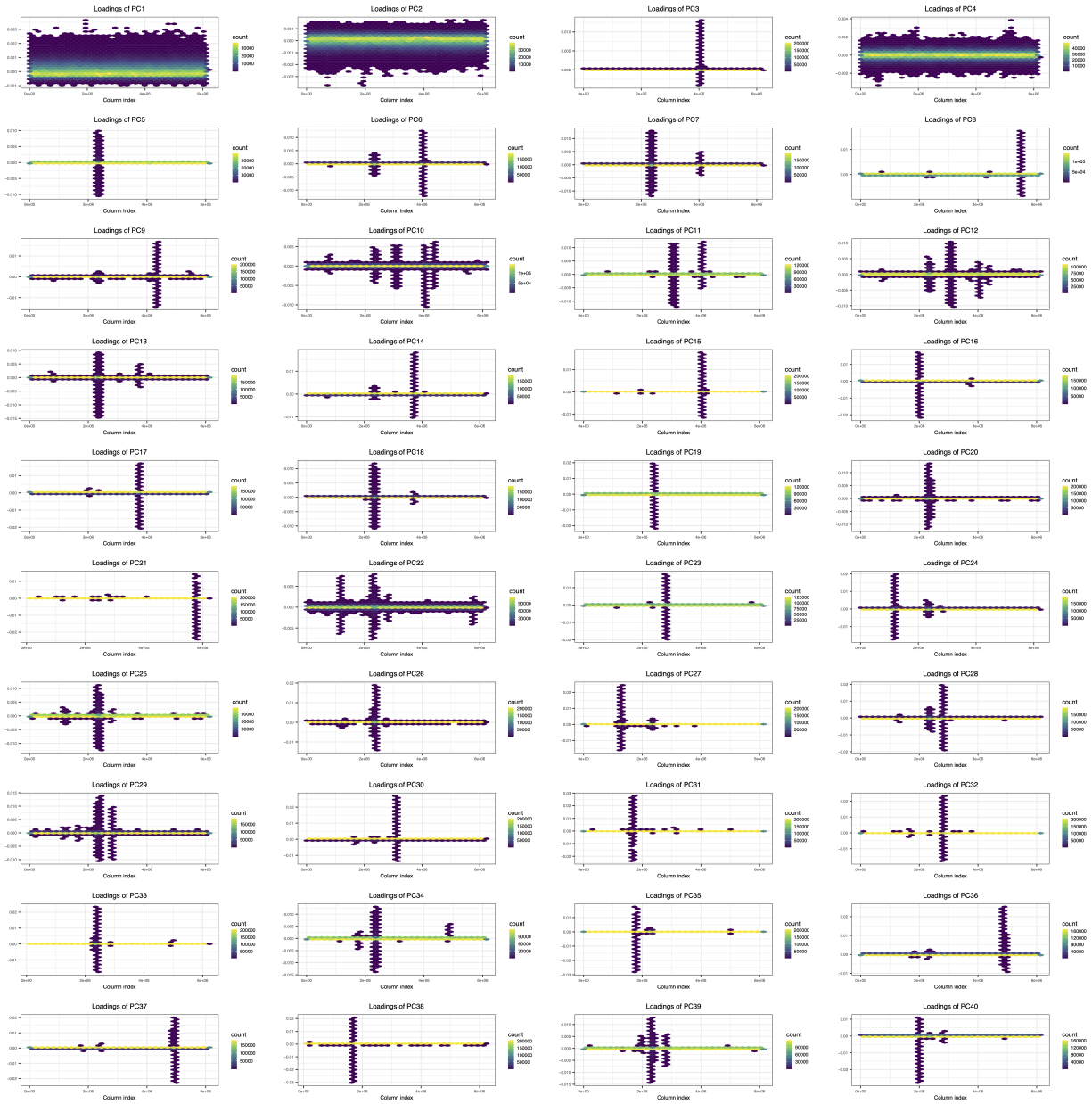


Figure S4: Top 40 PC loadings of the UK Biobank data with 487,409 individuals and 6,133,304 SNPs. Variants are ordered by chromosome and physical position, represented on the x-axis by their SNP number (Column index), and the value of loadings is represented on the y-axis. Points are hex-binned and colored by their density.

S2 Supplementary tables

Table S1: MEV against PCs by full-rank SVD as a function of top PCs for all populations from 1000 genomes with 2504 individuals and 5,766,022 SNPs.

PCs(k)	PCAoneA/FlashPCA2	PCAoneF	PCAoneH	FastPCA	TeraPCA	ProPCA
2	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
5	1.0000	1.0000	1.0000	1.0000	0.9999	1.0000
10	1.0000	1.0000	0.9990	1.0000	0.9981	0.9990
15	1.0000	1.0000	0.9994	1.0000	0.9922	0.9890
20	1.0000	1.0000	0.9997	1.0000	0.9494	0.9992
30	1.0000	0.9999	0.9976	0.9999	0.9984	0.9991
40	1.0000	0.9996	0.9741	0.9976	0.9777	0.9935

Table S2: MEV of top 20 PCs against PCAoneA for random subsets of UK biobank imputed data where the number of SNPs is fixed as 3,000,000 for all datasets. All programs run with $k = 20$ setting.

Dataset	Individuals	PCAoneF	PCAoneH	FastPCA/Plink2	TeraPCA	ProPCA
s1	10000	1.0000	0.9996	1.0000	0.9571	1.0000
s2	20000	1.0000	0.9997	1.0000	0.9632	1.0000
s3	40000	1.0000	0.9997	1.0000	0.9479	1.0000
s4	60000	1.0000	0.9998	1.0000	0.9789	1.0000
s5	80000	1.0000	0.9991	1.0000	0.9725	1.0000
s6	100000	1.0000	0.9995	1.0000	0.9503	1.0000

Table S3: MEV of top 20 PCs against PCAoneA for random subsets of UK biobank imputed data where the number of individuals is fixed as 100,000 for all datasets. All programs run with $k = 20$ setting.

Dataset	SNPs	PCAoneF	PCAoneH	FastPCA/Plink2	TeraPCA	ProPCA
s7	200000	1.0000	0.9997	1.0000	0.9860	1.0000
s8	400000	1.0000	0.9998	1.0000	0.9960	1.0000
s9	600000	1.0000	0.9999	1.0000	0.9982	1.0000
s10	800000	1.0000	1.0000	1.0000	0.9980	1.0000
s11	1000000	1.0000	0.9999	1.0000	0.9972	1.0000
s12	2000000	1.0000	0.9999	1.0000	0.9975	1.0000
s13	3000000	1.0000	0.9999	1.0000	0.9978	1.0000

Table S4: Number of passes over data (epochs) used by different programs for datasets in Table S2 and Table S3 given $k = 20$ setting.

Dataset	PCAoneF	PCAoneH	FastPCA/Plink2	TeraPCA	ProPCA
s1	5	13	40	44	44
s2	5	10	40	34	44
s3	5	10	40	32	44
s4	5	9	40	46	44
s5	5	11	40	40	44
s6	5	11	40	32	44
s7	7	11	40	30	44
s8	5	10	40	26	44
s9	5	10	40	30	44
s10	5	9	40	30	44
s11	5	9	40	28	44
s12	5	9	40	30	44
s13	5	9	40	40	44

Table S5: MEV against PCs estimated by full-rank SVD as a function of top PCs for scRNAs data with 12000 cells and 23771 genes. The default three iterations was used by OnlinePCA-Halko.

PCs	PCAoneA	PCAoneF	PCAoneH	OnlinePCA-Halko
2	1.0000000	1.0000000	1.0000000	0.9753135
3	1.0000000	1.0000000	1.0000000	0.9804843
4	1.0000000	1.0000000	1.0000000	0.9824888
5	1.0000000	1.0000000	1.0000000	0.9859905
6	1.0000000	1.0000000	1.0000000	0.9879956
7	1.0000000	1.0000000	1.0000000	0.9897004
8	1.0000000	1.0000000	1.0000000	0.9848932
9	1.0000000	1.0000000	1.0000000	0.9889352
10	1.0000000	1.0000000	1.0000000	0.9900309
11	1.0000000	1.0000000	1.0000000	0.9903505
12	1.0000000	1.0000000	1.0000000	0.9912341
13	1.0000000	1.0000000	1.0000000	0.9918151
14	1.0000000	1.0000000	1.0000000	0.9891819
15	1.0000000	1.0000000	0.9999999	0.9703988
16	1.0000000	1.0000000	0.9999998	0.9913479
17	1.0000000	0.9999999	0.9999996	0.9899419
18	1.0000000	0.9999999	0.9999987	0.9890437
19	1.0000000	0.9999997	0.9999976	0.9698647
20	1.0000000	0.9999995	0.9999940	0.9817112

Table S6: MEV against PCs estimated by PCAoneA as a function of top PCs for scRNAs data with 12000 cells and 23771 genes. The default three iterations was used by OnlinePCA-Halko.

PCs	FullSVD	PCAoneF	PCAoneH	OnlinePCA-Halko
2	1.0000000	1.0000000	1.0000000	0.9753135
3	1.0000000	1.0000000	1.0000000	0.9804843
4	1.0000000	1.0000000	1.0000000	0.9824888
5	1.0000000	1.0000000	1.0000000	0.9859906
6	1.0000000	1.0000000	1.0000000	0.9879956
7	1.0000000	1.0000000	1.0000000	0.9897004
8	1.0000000	1.0000000	1.0000000	0.9848932
9	1.0000000	1.0000000	1.0000000	0.9889352
10	1.0000000	1.0000000	1.0000000	0.9900309
11	1.0000000	1.0000000	1.0000000	0.9903505
12	1.0000000	1.0000000	1.0000000	0.9912342
13	1.0000000	1.0000000	1.0000000	0.9918151
14	1.0000000	1.0000000	1.0000000	0.9891819
15	1.0000000	1.0000000	0.9999999	0.9703989
16	1.0000000	1.0000000	0.9999998	0.9913479
17	1.0000000	0.9999999	0.9999996	0.9899419
18	1.0000000	0.9999999	0.9999987	0.9890437
19	1.0000000	0.9999997	0.9999977	0.9698647
20	1.0000000	0.9999995	0.9999940	0.9817112

Table S7: 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
PCAoneF	$1 - mev(V^{(i)}, V^{(i-1)}) < 1e-4$
PCAoneH	$1 - mev(V^{(i)}, V^{(i-1)}) < 1e-4$
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 S8: 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
PCAoneA/FlashPCA2	$O(nb)$	$O(nk^2)$
PCAoneF	$O(mb + 2ml + 3nl)$	$O(2nml)$
PCAoneH	$O(mb + 2ml + 3nl)$	$O(nml)$
TeraPCA	$O(mb + 2ml)$	$O(nml)$
FastPCA/Plink2	$O(nm + m(k+1)l)$	$O(nml)$
ProPCA	$O(nm)$	$O(nml/log_3(max(n,m))$