Estimation of site frequency spectra from low-coverage sequencing data using stochastic EM reduces overfitting, runtime, and memory usage

Rasmussen, Malthe Sebro; Garcia-Erill, Genís; Korneliussen, Thorfinn Sand; Wiuf, Carsten; Albrechtsen, Anders

Published in: Genetics

DOI: 10.1093/genetics/iyac148

Publication date: 2022

Document version
Publisher's PDF, also known as Version of record

Document license: Other

Estimation of site frequency spectra from low-coverage sequencing data using stochastic EM reduces overfitting, runtime, and memory usage

Malthe Sebro Rasmussen 1,*, Genís Garcia-Erill 2, Thorfinn Sand Korneliussen 3, Carsten Wiuf 3, Anders Albrechtsen 1

1Department of Biology, University of Copenhagen, 2200 København N, Denmark,
2Globe Institute, University of Copenhagen, 1350 København K, Denmark,
3Department of Mathematical Sciences, University of Copenhagen, 2100 København Ø, Denmark

*Corresponding author: Department of Biology, University of Copenhagen, Ole Maaløes Vej 5, 2200 København N, Denmark. Email: malthe.rasmussen@bio.ku.dk

Abstract

The site frequency spectrum is an important summary statistic in population genetics used for inference on demographic history and selection. However, estimation of the site frequency spectrum from called genotypes introduces bias when working with low-coverage sequencing data. Methods exist for addressing this issue but sometimes suffer from 2 problems. First, they can have very high computational demands, to the point that it may not be possible to run estimation for genome-scale data. Second, existing methods are prone to overfitting, especially for multidimensional site frequency spectrum estimation. In this article, we present a stochastic expectation–maximization algorithm for inferring the site frequency spectrum from NGS data that address these challenges. We show that this algorithm greatly reduces runtime and enables estimation with constant, trivial RAM usage. Furthermore, the algorithm reduces overfitting and thereby improves downstream inference. An implementation is available at github.com/malthesr/winsfs.

Keywords: site frequency spectrum; next-generation sequencing; expectation–maximization; genotype likelihoods; low-coverage data; demographic history

Introduction

The site frequency spectrum (SFS) is the joint distribution of allele frequencies among 1 or more populations, and it serves as an important summary statistic in population genetics. For instance, the SFS is sufficient for computing nucleotide diversity (Korneliussen et al. 2013), Fst (Bhatia et al. 2013), and f-statistics (Peter 2016). Furthermore, the SFS may be used for inferring demographic history (Marth et al. 2004; Gutenkunst et al. 2009; Excoffier et al. 2013) and selection (Tajima 1989; Fay and Wu 2000; Nielsen et al. 2005).

When working with high-quality data, it is usually straightforward to estimate the SFS from called genotypes. However, when genotype calls are uncertain, standard methods lead to significant bias in the estimated SFS (Nielsen et al. 2011), which propagates to downstream inference (Han et al. 2013). In particular, this situation arises when working with next-generation sequencing (NGS) data at low coverage and may be compounded by additional data-quality issues. Low-coverage NGS data are sometimes the only available option, for instance when working with ancient DNA (Olaide et al. 2019; Margaryan et al. 2020; van der Valk et al. 2021). Sequencing at low coverage is also a popular choice to reduce sequencing costs, since most of the key population genetics analysis remain possible with such data (Lou et al. 2021).

To estimate the SFS from low-coverage data, several methods have been proposed, which account for the genotype uncertainty in estimation of the SFS (Li 2011; Nielsen et al. 2011). These are based on finding the SFS that maximizes the data likelihood using numeric optimization. Two factors combine to create a computational challenge for such methods. First, to achieve an accurate estimate of the SFS, these methods usually require many iterations, each of which requires a full pass over the input data. Second, unlike most genetics analyses, the SFS cannot be based on only the small subset of the variable sites but must consider all sites. Taken together, this means that some summary of the full data must be held in RAM and iterated over many times. For genome-scale NGS data from more than a few dozen samples, or in more than 1 dimension, this is often not computationally feasible, as tens of hours of runtime and hundreds of gigabytes of RAM may be required. Current approaches for dealing with this issue restrict the analysis to fewer individuals and/or smaller regions of the genome (Sánchez-Barreiro et al. 2021), leading to less accurate results.

An additional problem with current methods is that they are prone to overfitting. In the multidimensional setting in particular, there is often very little information available for many of the entries in the frequency spectrum. Therefore, by considering the full data set, existing algorithms risk fitting noise, leading to estimates with poor generalizability.

In this article, we present a novel version of the stochastic expectation–maximization (EM) algorithm for estimation of the SFS...
from NGS data. In each pass through the data, this algorithm updates the SFS estimate multiple times in smaller blocks of sites. We show that for low-coverage whole genome sequencing (WGS) data, this algorithm requires only a few full passes over the data. This considerably reduces running time and means that it is possible to estimate the SFS using constant, negligible RAM usage by streaming data from disk. Moreover, by only considering smaller subsets of the data at a time, we show that this method reduces overfitting, which in turn leads to improved downstream inference.

Materials and methods

Estimation of the SFS from low-coverage sequencing data requires precomputing site allele frequency (SAF) likelihoods for each site, and these are based on genotype likelihoods. We begin by briefly reviewing these concepts.

Genotype likelihoods

Assume we have NGS data X sampled from K different populations (indexed by k), with Nk individuals in the kth population. Furthermore, say that we have M diallelic sites (indexed by m), so that G_mkn is the genotype of a diploid individual n at site m in population k, coding genotypes as the number of derived alleles. In the same way, we use X_mkn to refer to the sequencing data at this location.

We define the genotype likelihood p(X_mkn | G_mkn) as the probability of the data given a particular genotype. Genotype likelihoods form the basis of genotype calling and are calculated from aligned sequencing reads by various bioinformatic tools including bcftools/samtools (Li et al. 2009; Danecek et al. 2021), GATK (McKenna et al. 2010), and ANGSDD (Korneliussen et al. 2014), using slightly different models. For clarity, we outline the basic GATK model below, though the choice of model is not important for our purposes.

For D sequencing reads aligned to position m for individual n in population k, let b2 be the base call of the dth read. Assuming independence of base calls, we have

\[ p(X_{mkn} | G_{mkn} = g) = \prod_{d=1}^{D} p(b_d | G_{mkn} = g). \]  

If we consider the genotype as 2 alleles a1, a2 \in \{0, 1\} such that G_mkn = a1 + a2, then by random sampling of the parental alleles,

\[ p(b_d | G_{mkn} = g) = \frac{1}{2} p(b_d | a_1) + \frac{1}{2} p(b_d | a_2). \]  

In turn, this probability is modeled by

\[ p(b_d | a) = \begin{cases} \epsilon_d / 3 & \text{if } b_d \neq a \\ 1 - \epsilon_d & \text{else} \end{cases}, \]  

where \( \epsilon_d \) is the sequencing error probability associated with the dth base.

SAF likelihoods

Using genotype likelihoods, we can calculate SAF likelihoods, also sometimes known as sample allele frequency likelihoods. It is possible to think of the SAF likelihoods as the generalization of genotype likelihoods from individuals to populations: instead of asking about the probability of the data for 1 individual given a genotype, we ask about the probability of the data for a population given the sum of their derived alleles.

More formally, define the sum of derived alleles for population k at site m,

\[ Z_{mk} = \sum_{n=1}^{N_k} G_{mkn}, \]  

with \( Z_{mk} \in \{0, 1, \ldots, 2N_k\} \) corresponding to possible sample frequencies \{0, 1/2N_k, \ldots, 1\}. Now define the SAF likelihood for a single population k,

\[ p(X_{mkn} | Z_{mk} = j_k) = \sum_{g \in \{0,1\}\^{N_k}} p(g | Z_{mk} = j_k) \prod_{n=1}^{N_k} p(X_{mkn} | G_{mkn} = g_n), \]  

where \( X_{mkn} \) is the data for all individuals sampled in population k at site m, \( p(g | Z_{mk} = j_k) \) is the combinatorial probability of the genotype vector \( g = (g_1, \ldots, g_N) \) conditional on the sum of the genotypes being \( j_k \), and \( p(X_{mkn} | G_{mkn} = g_n) \) is a standard genotype likelihood. Using a dynamic programming algorithm, SAF likelihoods can be calculated from the genotype likelihoods of N individuals in \( O(N^p) \) time per site (Nielsen et al. 2012), and a linear time approximation has also been given (Han et al. 2014).

To extend this to the multidimensional SFS with K populations, let \( J = \times_{k=1}^{K} \{0, 1, \ldots, 2N_k\} \) be the set of possible derived allele count combinations across populations, let \( X_m \) be the data across all individuals in all populations at site m, and define \( Z_m = (Z_{m1}, \ldots, Z_{mk}) \in J \). Then

\[ p(X_m | Z_m) = \prod_{k=1}^{K} p(X_{mkn} | Z_{mk}), \]  

is the joint SAF likelihood for K populations.

Site frequency spectrum

Using the definition of \( J \) above, we define the SFS as a parameter \( \phi = \{\phi_j : j \in J\} \) such that \( \phi_j \) is the probability that \( Z_m = j \). That is,

\[ \phi_j = p(Z_m = j | \phi), \]  

for site m. That is, the SFS is the probability of a particular vector of derived allele sums at a site chosen at random. When genotypes are available, the SFS can be estimated simply by counting observed allele count combinations. When genotypes cannot be called, the standard approach is maximum likelihood estimation.

Assuming independence of sites, we write the likelihood function

\[ p(X | \phi) = \prod_{m=1}^{M} p(X_m | \phi) = \prod_{m=1}^{M} \sum_{j \in J} p(X_m | \phi, Z_m = j) p(Z_m = j | \phi) \]  

\[ = \prod_{m=1}^{M} \sum_{j \in J} p(X_m | Z_m = j) \phi_j, \]  

where \( X_m \) refers to all sequencing data for site m. Note that the likelihood can be expressed solely in terms of joint SAF likelihoods.
The maximum likelihood estimate \( \phi = \arg \max_{\phi} p(X \mid \phi) \) cannot be found analytically. Instead, \( \hat{\phi} \) is typically estimated using some iterative procedure such as BFGS (Nielsen et al. 2012) or an EM algorithm (Li 2011; Korneilussen et al. 2014), of which the latter has become the standard choice. An overview of the this algorithm is given below. For details and proof, see Supplementary Section 1.

**Standard EM algorithm**

Before optimization, we precompute the SAF likelihoods for all sites, populations, and possible sample frequencies. In addition, we make an arbitrary initial guess of the SFS \( \phi^{(0)} \). The EM algorithm then alternates between an E step and an M step.

The E step consists of computing posterior probabilities of derived allele counts conditional on the current SFS estimate,

\[
q_{mj}^{(t)} = \frac{p(Z_m = j \mid X_m, \phi^{(t)})}{\sum_{j' \in J} p(X_m \mid Z_m = j') \phi^{(t)}_{j'}},
\]

for all sites \( m \in \{1, \ldots, M\} \) and possible derived allele counts \( j \in J \). Note that this conditional posterior depends only on the current SFS estimate and the (joint) SAF likelihoods.

Using the result of the E step, the M step updates the estimate by setting

\[
\phi^{(t+1)}_j = \frac{\sum_{m=1}^M q_{mj}^{(t)}}{\sum_{m=1}^M \sum_{j' \in J} q_{mj}^{(t)}},
\]

for all \( j \in J \).

The EM algorithm guarantees a monotonically increasing likelihood of successive values of \( \phi^{(t)} \). The runtime of the algorithm is linear in the number of iterations required before convergence, with each iteration taking \( O(MN_k t) \) time. In practice, the standard implementation is realSFS (Nielsen et al. 2012) from the software suite ANGSD (Korneilussen et al. 2014), which uses a generic EM acceleration scheme (Varadhan and Roland 2008). The details of this acceleration will not be important in this context, so we omit the details.

**Window EM algorithm**

As in standard EM, we precompute all SAF likelihoods and make an arbitrary initial guess \( \phi^{(0)} \) of the SFS. In addition, we choose 2 hyperparameters \( B \) (the number of blocks) and \( W \) (the window size). Before starting optimization, all site indices are randomly assigned to one of \( B \) blocks \( B = \{B_1, \ldots, B_B\} \) with \( |B_b| = \lfloor M/B \rfloor \) for \( b < B \), and \( |B_B| = M \mod B \). The reason for doing so is simply to break patterns of linkage disequilibrium in particular blocks of input data, which will make the SFS within each block more similar to the global SFS. Blocks are nonoverlapping and exhaustive, so that \( \cup_{b=1}^B B_b = \{1, \ldots, M\} \) and \( \cap_{b=1}^B B_b = \emptyset \).

After this initialization, the window EM algorithm is defined as an iterative procedure that alternates between an E step and an M step, where the M step in turn is split into an M1 step and an M2 step.

The E step of the algorithm involves computing posteriors conditional on the current estimate of the SFS, much like standard EM. The difference is that we only process a single block of sites. Let \( f(t) = (t - 1) \mod B + 1 \), so that \( f(1 + xB) = 1, f(2 + xB) = 2, \ldots \) for \( x \geq 0 \). Then, at time step \( t \), we compute \( q_{mj}^{(t)} \) for all \( m \in B_{f(t+1)} \) and all possible derived allele counts \( j \in J \) using Equation (9).

In the M1 step, the \( \psi \) for the current block are used to give a block SFS estimate \( \psi^{(t)} \). This is analogous to the standard M step [Equation (10)], so that for each \( j \in J \),

\[
\psi_j^{(t+1)} = \frac{1}{|B_{f(t+1)}|} \sum_{m \in B_{f(t+1)}} q_{mj}^{(t)},
\]

These block estimates are then used in the M2 step to update the overall SFS estimate for each \( j \in J \),

\[
\psi_j^{(t+1)} = \frac{1}{W} \sum_{w \in W_t} \psi_j^{(w)} |B_w|,
\]

where \( W_t = \{f(t + 1 - w) \mid w \in \{0, \ldots, \min(t, W - 1)\}\} \) is the window of the \( W \) latest block indices at time \( t \). We use \( \circ \) to express equality under the common special case when either \( M/B = 0 \) or \( B \neq W_t \), so that there are no issues with blocks of unequal sizes in the current window. In this case, the M2 step simplifies to the mean of the past \( W \) block estimates.

Pseudo-code for window EM is given in Algorithm 1, and an illustration comparing window EM to standard EM is shown in Fig. 1.

In the below, we are interested in comparing standard EM and window EM. For clarity, we will use the term “epoch” to refer to a full pass through the data for either algorithm. In the case of standard EM, an epoch is simply a single iteration, for window EM, an epoch corresponds to \( B \) iterations.

**Convergence**

In the standard EM algorithm, the data log-likelihood [Equation (8)] can typically be evaluated with little computational overhead during the E step. Therefore, a common convergence criterion is to express equality under the common special case when either \( M/B = 0 \) or \( B \neq W_t \), so that there are no issues with blocks of unequal sizes in the current window. In this case, the M2 step simplifies to the mean of the past \( W \) block estimates.

Pseudo-code for window EM is given in Algorithm 1, and an illustration comparing window EM to standard EM is shown in Fig. 1.

In the below, we are interested in comparing standard EM and window EM. For clarity, we will use the term “epoch” to refer to a full pass through the data for either algorithm. In the case of standard EM, an epoch is simply a single iteration, for window EM, an epoch corresponds to \( B \) iterations.

**Algorithm 1 Window EM algorithm**

**Input**: (1) SAF likelihoods \( p(X_{mk} \mid Z_{mk} = j_k) \) for sites \( m \in \{1, \ldots, M\} \) and \( N_k \) individuals in each of populations \( k \in \{1, \ldots, K\} \), with \( j_k \in J = k \cdot \{1, \ldots, 2N_k\} \); (2) Random, nonoverlapping assignment of site indices from 1 to \( M \) into \( B \) blocks \( B_1, \ldots, B_B \); (3) Initial SFS estimate \( \phi^{(0)} \).

**Output**: Estimate \( \hat{\phi} \) of the \( K \)-dimensional SFS.

**Parameters**: Number of blocks \( B \), number of blocks per window \( W \).

\( t \leftarrow 0 \)

\( \text{while not converged do} \)

\( b_t \leftarrow t \mod B + 1 \) \> Block index

\( \text{for } m \in B_{b_t} \text{ do} \)

\( \text{for } j \in J \text{ do} \)

\( q_{mj}^{(t)} \leftarrow \frac{p(X_{mk} \mid Z_{mk} = j_k) \phi^{(t)}_{j_k}}{\sum_{j' \in J} p(X_{mk} \mid Z_{mk} = j'_k) \phi^{(t)}_{j'_k}} \) \> E step

\( W_t \leftarrow \{f(t + 1 - w) \mod B + 1 \mid w \in \{0, \ldots, \min(t, W - 1)\}\} \) \> Window indices

\( \psi_j^{(t+1)} \leftarrow \frac{1}{|B_{b_t}|} \sum_{m \in B_{b_t}} q_{mj}^{(t)} \psi_j^{(t)} \psi_j^{(w)} \) \> M1 step

\( \psi_j^{(t+1)} \leftarrow \frac{1}{W} \sum_{w \in W_t} \psi_j^{(w)} \psi_j^{(w)} \) \> M2 step

\( t \leftarrow t + 1 \)

\( \text{return } \hat{\phi}^{(t)} \)
based on the difference between the log-likelihood values of successive epochs. That is, let

\[ L_t = \frac{1}{M} \sum_{m=1}^{M} \log p(X | \hat{\phi}_m(t)) , \]  

(13)

and convergence is reached when \( L_{t+1} - L_t < \delta \), for some tolerance \( \delta \) decided ahead of time.

For window EM, the same does not apply, since no full E step is ever taken. However, the likelihood for each block can be calculated cheaply during each block E step. Therefore, we define for epoch \( e \in \{1, 2, \ldots\} \),

\[ L'_e = \frac{1}{B} \sum_{b=1}^{B} \sum_{m \in B_b} \log p(X_m | \hat{\phi}_{b+\delta}^{(e-b)}) , \]  

(14)

that is, the sum of log-likelihoods of SFS estimates used over the past epoch, each evaluated in the block for which they were used in a block E step, normalized by block size for convenience.

### Results

To test the window EM algorithm, we implemented it in the `winsfs` program, available at github.com/malthesr/winsfs. We compare `winsfs` to `realSFS`, which implements the standard EM algorithm and serves as the current state of art. We adopt 2 complementary approaches for evaluating performance of `winsfs`. First, we use 2 different real-world WGS data sets to compare `winsfs` to `realSFS`. Second, we use simulated data to validate `winsfs` under conditions of known truth across a range of data qualities and sample sizes.

### Real-world data sets

We tested `winsfs` and `realSFS` on 2 real-world WGS data sets of very different quality as described below. An overview is shown in Table 1.
We first analyzed 10 random individuals from each of the YRI (Yoruba Nigerian) and CEU (Europeans in Utah) populations from the 1000 Genomes Project (The 1000 Genomes Project Consortium 2015). This human data were sequenced to 3× - 8× coverage and mapped to the high-quality human reference genome. We created SAF files using ANGSD (Korneliussen et al. 2014) requiring minimum base and mapping quality 30 and polarizing the spectrum using the chimpanzee as an outgroup. We then split these input data into test and training data, such that the first half of each autosome was assigned to the training set and the second half to the test set. The resulting training data set contains 1.17 × 10⁹ sites for both YRI and CEU, while the test data set contains 1.35 × 10⁹ sites for both. Training set depth distributions for each individual are shown in Supplementary Fig. 1.

We also analyze a data set of much lower quality consisting of 12 and 8 individuals from 2 impala populations that we refer to as “Maasai Mara” and “Shangani,” respectively, based on their sampling locations. These populations were sequenced to only 1× - 3× with the addition of a single high-depth sample in each population (see Supplementary Fig. 2). The data were mapped to a very fragmented assembly, and then, we split the data into training and test sets just as for the human data. However, due to the low-quality assembly, we analyzed only sites on contigs larger than 100 kb and filtered sites based on depth outliers, excess heterozygosity, mappability, and repeat regions. We polarized using the impala reference itself. This process is meant to mirror a realistic workflow for working with low-quality data from a nonmodel organism. The impala input data end up some-what smaller than the human data set, with approximately 6.3 × 10⁸ sites in both test and training data sets.

Broadly, the human data are meant to exemplify medium-quality data with coverage toward the lower end, but with no other significant issues. The impala data, on the other hand, represent low-quality data: not only the coverage low and fewer sites are available, but also the impala reference genome is poor quality with 7,811 contigs greater than 100 kb and mₚₛ = 3.4 × 10⁻⁵ (i.e. 50% of the assembly bases lie on contigs of this size or greater). This serves to introduce further noise in the mapping process, which amplifies the overall data uncertainty. Finally, the impala populations are more distinct, with Fₛₚ ≈ 0.24 compared to 0.13 between the human populations. As we will see below, this creates additional challenges for estimation of the 2-dimensional SFS.

**Estimation**

Using the training data sets, we estimated the 1-dimensional SFS for YRI and Maasai Mara, as well as the 2-dimensional SFS for CEU/YRI and Shangani/Maasai Mara. We ran winsfs for 500 epochs using a fixed number of blocks \( B = 500 \) and window sizes \( W \in \{100, 250, 500\} \). We will focus on the setting with window size \( W = 100 \). For convenience, we introduce the notation \( \text{winsfs}_{100} \) to refer to \( \text{winsfs} \) with hyperparameter settings \( B = 500 \) and \( W = 100 \). We return to the topic of hyperparameter settings below.

To compare, we ran realSFS using default settings, except allowing it to run for a maximum of 500 epochs rather than the default 100. We will still take the 100 epochs cutoff to mark convergence, if it has not occurred by other criteria before then, but results past 100 will be shown in places.

In each case, we evaluated the full log-likelihood (Equation (8)) of the estimates after each epoch on both the training and test data sets. In addition, we computed various summary statistics from the estimates after each epoch. For details, see Supplementary Section 2.

**One-dimensional SFS**

Main results for the 1-dimensional estimates are shown in Fig. 2.

For the human YRI population, we find that a single epoch of \( \text{winsfs}_{100} \) produces an estimate of the SFS that is visually indistinguishable from the converged estimate of realSFS at 39 epochs (Fig. 2a). Train and test set log-likelihoods (Supplementary Fig. 3) confirm that the likelihood at this point is very marginally lower for \( \text{winsfs}_{100} \) than the last realSFS. By increasing the window size to 250 or 500, we get test log-likelihood values equal to or above those achieved by realSFS, and still within the first 5 epochs.

As an example of a summary statistic derived from the 1-dimensional SFS, Fig. 2b shows that \( \text{winsfs}_{100} \) finds an estimate of Tajima’s \( \theta \) that is very near to the final realSFS, with a difference on the order of 1 × 10⁻⁶. Increasing the window size removes this difference at the cost of a few more epochs.

In the case of Maasai Mara, realSFS runs for the 500 epochs, so we take epoch 100 to mark convergence. On these data, \( \text{winsfs}_{100} \) requires 2 epochs to give a good estimate of the SFS, as shown in Fig. 2c. Some subtle differences relative to the realSFS results remain, however, especially at the middle frequencies: the realSFS estimate exhibits a “wobble” such that even bins are consistently higher than odd bins. Such a pattern is not biologically plausible and is not seen in the \( \text{winsfs} \) spectrum.

Supplementary Fig. 4 shows train and test log-likelihood data for Maasai Mara, which again support the conclusions drawn from looking at the estimates themselves. In theory, we expect that the test log-likelihood should be adversely impacted by the realSFS “wobble” pattern. In practice, however, with more than 99.5% fixed sites, the fixed end of the spectrum dominate the likelihood to the extent that the effect is not visible. We return to this point below.

Finally, Fig. 2d shows that Tajima’s \( \theta \) is likewise well-estimated by 1 or 2 epochs of \( \text{winsfs}_{100} \) on the impala data.

**Two-dimensional SFS**

Overall results for the joint spectra are seen in Fig. 3.

On the human data, \( \text{winsfs}_{100} \) takes a single epoch for an estimate of the SFS that is near-identical to realSFS at convergence after 93 epochs. Looking at the log-likelihood results, it is notable

<table>
<thead>
<tr>
<th>Table 1. Overview of the input training data.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Population</td>
</tr>
<tr>
<td>------------</td>
</tr>
<tr>
<td>Human</td>
</tr>
<tr>
<td>YRI</td>
</tr>
<tr>
<td>CEU</td>
</tr>
<tr>
<td>Impala</td>
</tr>
<tr>
<td>Maasai Mara</td>
</tr>
<tr>
<td>Shangani</td>
</tr>
</tbody>
</table>

We return to the topic of hyperparameter settings below.

To compare, we ran realSFS using default settings, except allowing it to run for a maximum of 500 epochs rather than the default 100. We will still take the 100 epochs cutoff to mark convergence, if it has not occurred by other criteria before then, but results past 100 will be shown in places.

In each case, we evaluated the full log-likelihood (Equation (8)) of the estimates after each epoch on both the training and test data sets. In addition, we computed various summary statistics from the estimates after each epoch. For details, see Supplementary Section 2.

**One-dimensional SFS**

Main results for the 1-dimensional estimates are shown in Fig. 2.

For the human YRI population, we find that a single epoch of \( \text{winsfs}_{100} \) produces an estimate of the SFS that is visually indistinguishable from the converged estimate of realSFS at 39 epochs (Fig. 2a). Train and test set log-likelihoods (Supplementary Fig. 3) confirm that the likelihood at this point is only very marginally lower for \( \text{winsfs}_{100} \) than the last realSFS. By increasing the window size to 250 or 500, we get test log-likelihood values equal to or above those achieved by realSFS, and still within the first 5 epochs.

As an example of a summary statistic derived from the 1-dimensional SFS, Fig. 2b shows that \( \text{winsfs}_{100} \) finds an estimate of Tajima’s \( \theta \) that is very near to the final realSFS, with a difference on the order of 1 × 10⁻⁶. Increasing the window size removes this difference at the cost of a few more epochs.

In the case of Maasai Mara, realSFS runs for the 500 epochs, so we take epoch 100 to mark convergence. On these data, \( \text{winsfs}_{100} \) requires 2 epochs to give a good estimate of the SFS, as shown in Fig. 2c. Some subtle differences relative to the realSFS results remain, however, especially at the middle frequencies: the realSFS estimate exhibits a “wobble” such that even bins are consistently higher than odd bins. Such a pattern is not biologically plausible and is not seen in the \( \text{winsfs} \) spectrum.

Supplementary Fig. 4 shows train and test log-likelihood data for Maasai Mara, which again support the conclusions drawn from looking at the estimates themselves. In theory, we expect that the test log-likelihood should be adversely impacted by the realSFS “wobble” pattern. In practice, however, with more than 99.5% fixed sites, the fixed end of the spectrum dominate the likelihood to the extent that the effect is not visible. We return to this point below.

Finally, Fig. 2d shows that Tajima’s \( \theta \) is likewise well-estimated by 1 or 2 epochs of \( \text{winsfs}_{100} \) on the impala data.

**Two-dimensional SFS**

Overall results for the joint spectra are seen in Fig. 3.

On the human data, \( \text{winsfs}_{100} \) takes a single epoch for an estimate of the SFS that is near-identical to realSFS at convergence after 93 epochs. Looking at the log-likelihood results, it is notable
that while realSFS does better than winsfs when evaluated on the training data (Fig. 3b), the picture is reversed when evaluated on the test data (Fig. 3c). In fact, all winsfs hyperparameter settings achieved better test log-likelihood values in the first 10 epochs than achieved by realSFS at convergence. This is likely caused by a faint “checkerboard” pattern in the realSFS estimate due to overfitting, as we expect the spectrum to be smooth. We note that both realSFS and winsfs preserve an excess of sites where all individuals are heterozygous, corresponding to the peak in the center of the spectrum. This is a known issue with this data set (Meisner and Albrechtsen 2019), likely caused by paralogs in the mapping process. It is an artifact that can be removed by filtering the data before SAF calculation, which we have not done here. Given this choice, it is to be expected that this peak remains.

In 2 dimensions, we compute both Hudson’s $F_2$ (Fig. 3d) and the $f_2$-statistic (Supplementary Fig. 5) from SFS estimates after all epochs, and we note similar patterns for these as we have seen before: 1 epoch of winsfs gives an estimate of the summary statistic that is almost identical to the final realSFS estimate.

For the impalas, winsfs requires 2 epochs for a good estimate of the spectrum, while realSFS again does not report convergence within the first 100. What is immediately striking about the impala results, however, is that the checkerboard pattern is very pronounced for realSFS, and again absent for winsfs (Fig. 3e). The problem for realSFS is likely exacerbated by 2 factors: first, the sequencing depth is lower, increasing the uncertainty, and second, the relatively high divergence of the impala populations pushes most of the mass in the spectrum towards the edges. Together, this means that very little information is available for most of the estimated parameters. It appears that realSFS therefore ends up overfitting to the particularities of the training data at these bins.

This is also reflected in the difference between train and test log-likelihood (Fig. 3f and g). Like in the case of the human data, the SFS estimates by winsfs performs better on the test data compared to realSFS, while realSFS performs the best on the training data. On the test data, all winsfs settings again reach log-likelihood values comparable to or better than realSFS in few epochs. However, the differences between realSFS and winsfs remain relatively small in terms of log-likelihood, even on the test set. This is somewhat surprising, given the marked checkerboarding in the spectrum itself. Again, we attribute this to the fact that the log-likelihood is dominated by all the mass
lying in or around the zero–zero bin. We expect, therefore, that methods that rely on the “interior” of the SFS should do better when using winsfs, compared to realSFS.

Before turning to test this prediction, we briefly note that $F_{st}$ (Fig. 3h) and the $f_2$-statistic (Supplementary Fig. 5) are also adequately estimated for the impalas by winsfs in 1 epoch.
Demographic inference

All the SFS-derived summary statistics considered so far are heavily influenced by the bins with the fixed allele bins (i.e. count 0 or 2N, in all populations), or they are sums of alternating frequency bins. In either case, this serves to mask issues with checkerboard areas of the SFS in the lower-frequency bins. However, this will not be the case for downstream methods that rely on the shape of the spectrum in more detail.

To illustrate, we present a small case study of inferring the demographic history of the impala populations using the *daulh* (Gutenkunst et al. 2009) software with the estimated impala spectra shown in Fig. 3e, though folded due to the lack of an outgroup for proper polarization. Briefly, based on an estimated SFS and a user-specified demographic model, *daulh* fits a model SFS based on the demographic parameters so as to maximize the likelihood of these parameters. Our approach was to fit a simple demographic model for the Shangani and Maasai Mara populations and then gradually add parameters to the model as required based on the residuals of the input and model spectra. We take this to be representative of a typical workflow for demographic inference.

For each successive demographic model (Portik et al. 2017), we ran *daulh* on the folded spectra by performing 100 independent optimization runs from random starting parameters and checking for convergence by requiring the top 3 results to be within 5 log-likelihoods units of each other. If the optimization did not converge, we did additional optimization runs until either they converged or 500 independent runs were reached without likelihood convergence. In that case, we inspected the results for the top runs, to assess whether they were reliably reaching similar estimates and likelihoods. Results are shown in Fig. 4.

The first, basic model assumes that the populations have had constant populations sizes and a symmetric migration rate since diverging. The parameter estimates based on *realSFS* and *windsf* are similar, though the *windsf* model fit has significantly higher log-likelihood (Fig. 4a). However, when inspecting the residuals in Fig. 4b, the *realSFS* residuals suffer from a heavy checkerboard pattern, making it hard to distinguish noise from model misspecification. In contrast, the *windsf* residuals clearly show areas of the spectrum where the model poorly fits the data.

In particular, the residuals along the very edge of the spectrum suggest that a symmetric migration rate is not appropriate. Therefore, we fit a second model with asymmetric migration (Fig. 4c). Now *daulh* finds migration rates from Shangani to Maasai Mara an order of magnitude higher than vice versa. The results for *windsf* (Fig. 4d) show improved residuals, while the *realSFS* residuals remain hard to interpret.

Finally, an area of positive residuals in the fixed and rare-variant end of the Shangani spectrum suggests that this population has recently undergone a significant bottleneck. Therefore, the third model allows for an instantaneous size change in each of the impala populations (Fig. 4e). At this point, the *windsf* residuals (Fig. 4f) are negligible, suggesting that no more parameters should be added to the model. Once again, however, the *realSFS* residuals leave us uncertain whether further model extensions are required.

When looking at the final model fits, the *daulh* parameter estimates from *realSFS* and *windsf* also start to differ slightly. In several instances, estimates disagree by about 50%, and the log-likelihood remains much higher for *windsf*, with a difference of 45,000 log-likelihood units to *realSFS*. In addition, we confirmed that the log-likelihood of the data in the original test SAF files given the SFS fitted by *daulh* is higher for *windsf* (-8.08 x 10^6) than for *realSFS* (-8.38 x 10^6). We stress, however, that we would have likely never found the appropriate model without using *windsf*, since the interpretation of the *realSFS* results is difficult. In relation to this point, we note that the final model results in considerably different estimates for parameters of biological interest, such as split times and recent population sizes, relative to the initial model. We also find that the last model is supported by the literature: previous genetic and fossil evidence suggests extant common impala populations derive from a refugium in Southern Africa that subsequently colonized East Africa in the middle-to-late Pleistocene (Lorenzen et al. 2006, 2012; Faith et al. 2013). This is broadly consistent with the estimated split time, and the reduction in population size in East African populations as they colonized the new habitat. The difference in effective population size between the southern Shangani population and the eastern Maasai Mara was previously also found using microsatellite data (Lorenzen et al. 2006).

Simulations

To validate these findings in conditions with a known SFS, we ran simulations using *msprime* (Baumdicker et al. 2021) and *tskit* (Kelleher et al. 2018). Briefly, we simulated 2 populations, which we simply refer to as A and B. Populations A and B diverged 10,000 generations ago and both have effective populations sizes of 10,000 individuals, except for a period of 1,000 generations after the split, during which time B went through a bottleneck of size 1,000. We simulated 22 independent chromosomes of 10 Mb for a total genome size of 220 Mb, using a mutation rate of 2.5 x 10^-8 and a uniform recombination rate of 1 x 10^-8. To explore the consequences of varying sample sizes, we sampled 5, 10, or 20 individuals from the 2 populations. For each of these 3 scenarios, we calculated the true SFS from the resulting genotypes (shown in Supplementary Fig. 6).

Using the true genotypes as input, we simulated the effects of NGS sequencing with error for both the variable and invariable sites. At every position in the genome, including the monomorphic sites, we sample D ~ Poisson(λ) bases and introduce errors with a constant rate of ε = 0.002 independently for each base. We calculated genotype likelihoods according to the GATK model outlined in Equations (1)–(3) and output GLF files. Using these, we created SAF files for A and B with no further filtering using *angsdf*. The mean depth λ was set to either 2, 4, or 8 to investigate the performance of *windsf* at different sequencing depths. This results in a grid of 3 x 3 simulated NGS data sets with 3 different sample sizes and 3 different mean depth values.

From the simulated SAF files, we ran *windsf* and *realSFS* as above to generate the 2-dimensional SFS, except for a maximum of 100 epochs. For each method and each epoch until convergence, we calculated the log-likelihood for the corresponding SFS ^ SFS (\(\hat{\phi}\)):

\[
\log p(\phi \mid ^ SFS (\\hat{\phi})) = \log \prod_{j \in J} \phi_j^{M_j} = \sum_{j \in J} M_j \log \phi_j
\]

where \(\phi_j\) is the observed true SFS and \(M_j\) is the total number of sites. Figure 5 shows how the log-likelihood evolves over epochs for *windsf* (W ∈ {100,250,500}) and *realSFS* for sample sizes \(N_j \in \{5,10,20\}\) and simulated mean depths \(\lambda \in \{2,4,8\}\). We observe that at a mean depth of 2, *windsf* outperforms *realSFS* by a significant margin both in terms of speed and the final log-likelihood. At mean depth 4, the *windsf* remains much faster.
and still achieves meaningfully better log-likelihoods, especially at higher sample sizes. Finally, at mean depth 8, winsfs,100 still converges 5–10 times faster than realSFS (measured in epochs), but the methods provide estimates of similar quality.

The estimated spectra for realSFS and winsfs,100 at their default stopping points are shown in Supplementary Figs. 7 and 8, respectively. These confirm that the spectra on the whole are well-estimated by winsfs,100 as compared to the true SFS (Supplementary Fig. 6). Moreover, we again observe that realSFS introduces a checkerboard pattern in the low-information part of the spectrum at $2\pi/C^2=4\pi/C^0$, which is not present in the true spectrum, and which is not inferred by winsfs. The pattern is more...
pronounced at higher sample sizes. This supports the hypothesis that realSFS tends to overfit in situations where many parameters must be inferred with little information.

**Peak simulations**

The averaging of block estimates in the window EM algorithm appears to induce a certain “smoothing” of the spectrum at low depth. This smoothing effect is implicit in the sense of being nowhere explicitly modeled, and each parameter is estimated independently. Nevertheless, this observation may give rise to a concern that winsfs, unlike the maximum likelihood estimate from realSFS, might remove true abrupt peaks in the SFS.

To investigate, we modified the demographic simulation with sample size 20 described above in the following way. In each of 7 arbitrarily chosen bins near to the center of the SFS, we artificially spiked 10,000 counts into the true spectrum after running the demographic simulations (Supplementary Fig. 9). This represents an increase of 30- to 40 times relative to the original count and the neighboring cells. Based on this altered spectrum, we simulated sequencing data for depth 2, 4, or 8, created SAF files, and ran realSFS and winsfs 100 as before. The residuals of the realSFS and winsfs estimates are shown in Supplementary Figs. 10 and 11, respectively. In this fairly extreme scenario, the spectra inferred by both winsfs and realSFS appear to have a small but noticeable downwards bias in the peak region at 2x and 4x. However, compared to realSFS, winsfs has smaller residuals in all scenarios, and the apparent bias is inversely correlated with depth. These results confirm that using the window
EM algorithm does not lead to excess flattening of SFS peaks compared with the maximum likelihood estimate from the standard EM algorithm.

Hyperparameters
The window EM algorithm requires hyperparameter settings for \( B \) and \( W \). Moreover, it requires a choice of stopping criterion. For ease of use, the \texttt{winsfs} software ships with defaults for these settings, and we briefly describe these.

We expect that the choice of \( B \) is less important than the term \( W/B \), which governs the fraction of data that is directly considered in any one update step. Having analyzed input data varying in size from 220 Mb (simulations) to 1.17 GB (human data), we find that fixing \( B = 500 \) works fine as a default across a wide range of input sizes. Therefore, the more interesting question is how to set the window size. In theory, there should be a tradeoff between speed of convergence and accuracy of results, where lower window size favors the former and higher window size the latter. However, in practice, based on our results, we have not seen evidence that using \( W = 500 \) over \( W = 100 \) leads to significantly better inference. On the other hand, the lower window size has significantly faster convergence. Based on this, we feel that window size of 100 makes for the best general default. By default, the \texttt{winsfs} software uses \( B = 500 \) blocks and a window size \( W = 100 \).

As for stopping, \texttt{winsfs} implements the criterion based on differences \( \delta \) in \( L_1 \) [Equation (14)] over successive epochs. Based on the initial analysis of the human and impala data, we chose \( \delta = 10^{-4} \) (see Supplementary Fig. 12) as the default value and used the simulations to validate this choice. Figure 5 shows the point at which stopping occurs, which is generally around the maximum log-likelihood as desired.

Streaming
In the main usage mode, precalculated SAF likelihoods are read into RAM, as in realSFS. However, it is also possible to run \texttt{winsfs} while keeping the data on disk and streaming through the intersecting sites in the SAF files. We refer to this as a “streaming mode.”

Since the window EM algorithm requires randomly shuffling the input data, a preparation step is required in which SAF likelihoods are (jointly) shuffled into a new file. We wish to avoid loading the data into RAM to perform a shuffle, and we also do not want multiple intermediate writes to disk. To our knowledge, it is not possible to perform a true shuffle of the input data within these constraints. Instead, since we are only interested in shuffling for the purposes of breaking up blocks of LD, we perform a pseudo-shuffle according to the following scheme. We preallocate a file with space for exactly \( M \) intersecting sites in the input data. This file is then split into \( S \) contiguous sections of roughly equal size, and we then assign input site with index \( m \in \{1, \ldots, M\} \) to position \( \lfloor (m + 1)/S \rfloor \) in section \( (m + 1)\%S + 1 \), where \% is the remainder operation. That is, the first \( S \) sites in the input end up in the first positions of each section, and the next \( S \) sites in the input end up in the second positions of each section, and so on. This operation can be performed with constant memory, without intermediate writes to disk, and has the benefit of being reversible.

After preparing the pseudo-shuffled file, \texttt{winsfs} can be run exactly as in the main mode. To confirm that this pseudo-shuffle is sufficient for the purposes of the window EM algorithm, we ran 10 epochs of \texttt{winsfs} in the streaming mode for the impala and human data sets in both 1 and 2 dimensions. After each epoch, we calculated the log-likelihood of the resulting SFS and compared them to the log-likelihood obtained by running in main mode above. The results are shown in Supplementary Fig. 13 and show that streaming mode yields comparable results to the main, in-RAM usage: the likelihood differs slightly, but is neither systematically better or worse.

Benchmark
To assess its performance characteristics, we benchmarked \texttt{winsfs} in both the main mode and the streaming mode as well as realSFS on the impala data. For each of the 3, we ran estimation until convergence, as well as until various epochs before then, collecting benchmark results using Snakemake (Koster and Rahmann 2012). Both realSFS and \texttt{winsfs} were given 20 cores. Results are shown in Fig. 6. In terms of runtime, we find that running \texttt{winsfs} in RAM is significantly faster than realSFS (Fig. 6a). This is true in part because \texttt{winsfs} requires fewer epochs, but also since \texttt{winsfs} runs faster than realSFS epoch by epoch. As expected, when switching \texttt{winsfs} to the streaming mode,
runtime suffers as epochs increase. However, taking the number of epochs required for convergence into account, and streaming winsfs remains competitive with realSFS, even when including the initial overhead to shuffle SAF likelihoods on disk.

Looking at memory consumption, streaming winsfs has a trivial peak memory usage of 10 MB, including the initial pseudo-shuffle. In comparison, when reading data into RAM, realSFS and winsfs require 137 and 107 GB, respectively, even on the fairly small impala data set.

Discussion

We have presented the window EM algorithm for inferring the SFS from low-depth data, as well as the winsfs implementation of this algorithm. The window EM algorithm updates SFS estimates in smaller blocks of sites and averages these block estimates in larger windows. We have argued that this approach has 3 related advantages relative to current methods. First, by updating more often, convergence happens 1–2 orders of magnitude faster. Due to the window averaging, this improvement in convergence times does not occur at the cost of stability. Second, due to the fast convergence, it is feasible to run the window EM algorithm out of memory. This brings the memory requirements of the algorithm from hundreds of gigabytes of RAM to virtually nothing. Third, by optimizing over different subsets of the data in each iteration, the algorithm is prevented from overfitting to the input data. In practice, this means that we get biologically more plausible spectra.

On this last point, it is worth emphasizing that while winsfs appears to have the effect of smoothing the spectrum in a beneficial way, this smoothing effect is entirely implicit. That is, it is nowhere explicitly modeled that each estimated bin should be similar to neighboring bins to avoid checkerboard patterns. Rather, the apparent smoothing emerges because winsfs mitigates some of the issues with overfitting that may otherwise manifest as a checkerboard pattern. As shown in the simulations, winsfs does not remove true peaks in the SFS. In the broader setting of stochastic optimization, window EM is in this way related to forms of Polyak-Ruppert iterate averaging schemes as used in stochastic gradient methods (Ruppert 1988; Polyak and Juditsky 1992), variants of which have also been shown to control variance and induce regularization (Jain et al. 2018; Neu and Rosasco 2018), similar to what we have observed here.

Within the EM literature, window EM is prima facie quite similar in spirit to other versions of the stochastic EM algorithm (Neal and Hinton 1998; Sato and Ishii 2000; Cappé and Moulines 2009; Liang and Klein 2009; Chen et al. 2018). They too work on smaller blocks and seek some way of controlling stability in how the block estimate \( \hat{\theta} \) is incorporated in the overall estimate \( \hat{\theta} \). Typically, this involves an update of the form \( \gamma \hat{\theta} + (1 - \gamma) \phi \) for some weight \( \gamma \) decaying as a function of iteration \( t \). During initial experimentation, we empirically found that such methods tended to increase the noise in the spectrum, rather than reduce it. This problem likely arises because estimating the multidimensional SFS requires estimating many parameters for which very little information is available in any 1 batch. Therefore, by having an update step involving only the current estimate and a single, small batch of sites, significant noise is introduced in the low-density part of the spectrum. In contrast, the window EM approach still optimizes over smaller batches for speed but actually considers large amounts of data in the update step by summing the entire window of batch estimates, thereby decreasing the noise.

For SFS inference specifically, prior work exists to improve estimation for low-depth sequencing data. For example, it has been proposed to “band” SAF likelihoods to make estimation scale better in the number of sampled individuals (Han et al. 2014; Mas-Sandoval et al. 2022). Briefly, the idea is that at each site, all the mass in the SAF likelihood tends to be concentrated in a small band around the most likely sample frequency and downstream inference can be adequately carried out by only propagating this band and setting all others to zero. By doing so, runtime and RAM can be saved by simply ignoring all the zero bins outside the chosen band. We note that such ideas are orthogonal to the work presented here, since they are concerned with the representation of the input data, and thereby indirectly modify all downstream optimization methods. Future work on winsfs may involve the ability to run from banded SAF likelihoods. This will be important with large sample sizes, in the hundreds of individuals.

Others have focused on the implementation details of the EM algorithm, for instance using GPU acceleration (Lu et al. 2012). Such efforts still have the typical high memory requirements and do not address the overfitting displayed by the standard EM algorithm. Moreover, we find that the presented algorithmic improvements, combined with an efficient implementation, serve to make winsfs more than competitive with such efforts in terms of runtime. Indeed, with winsfs converging in-memory in less than an hour on genome-scale data, runtime is no longer a significant bottleneck for SFS estimation.

We emphasize, however, that the window EM algorithm and winsfs are unlikely to yield any meaningful benefits with sequencing data at above around \( 10^{\times-12} \times \) coverage. With such data, better inference of the SFS will be obtained by estimation directly from genotype calls with appropriate filters. Nevertheless, efficient and robust methods remain important for low-coverage data. This is partly because low-coverage data may sometimes be the only option, for example when working with ancient DNA. Also, such methods allow intentionally sequencing at lower coverage, decreasing the sequencing cost per individual.

In addition, we do not expect winsfs to perform better than realSFS when data are not available for many sites (e.g. <100 Mb) due to the fact that winsfs only uses parts of the available data directly in the final estimation.

Finally, improvements in the SFS estimates by winsfs are unlikely to be significant for simple summary statistics like \( \theta, F_{ST}, \) or \( f \)-statistics. For such purposes, winsfs simply produces results similar to realSFS, although much faster. However, as the number of dimensions and samples increase, and as sequencing depth decreases, overfitting will start to influence the low-density bins of the spectrum. Where this information is used downstream, winsfs will lead to better and more interpretable results and can potentially help solve commonly known biases in parameter estimates arising from model misspecification (Morigliano et al. 2021). We have seen this in the \( \theta_{0} \) case study, but we believe that the same would be true of other popular demographic inference frameworks including fastsimcoal (Excoffier and Foll 2011; Excoffier et al. 2013), moments (Jougangous et al. 2017), and momi (Kamm et al. 2017). It may also be significant for other methods for complex inference from the multidimensional spectrum, including inference of fitness effects using \( f \)-statistic (Kim et al. 2017; Huang et al. 2021) or introgression using \( D_{S} \) (Martin and Amos 2020), though we have not explored these methods.

Data availability

The human data analyzed are part of the 1000 Genomes (The 1000 Genomes Project Consortium 2015) phase 3 low depth sequencing data. Alignments have been made available by the
1000G project and can be accessed at ftp.1000genomes.ebi.ac.uk/vol1/ftp/phase3/. The impala data have been made available via the SRA with accession PRJNA862915. Analysis and plotting code, as well as the cleaned data corresponding to the final results, are available at github.com/malthesr/window and the winsfs software itself at github.com/malthesr/winsfs. Supplemental material is available at GENETICS online.

Funding

AA, CW, and MSR are supported by the Independent Research Fund Denmark (grant numbers: 8021-00360B and 0135-00211B) and the University of Copenhagen through the Data+ initiative. GGE is supported by the Independent Research Fund Denmark (grant number: 8049-00098B). TSK is funded by a Carlsberg Foundation Young Researcher Fellowship awarded by the Carlsberg Foundation in 2019 (CF19-0712).

Conflicts of interest

None declared.

Literature cited


Mas-Sandoval A, Pope NS, Nielsen KN, Altinkaya I, Fumagalli M, Korneliussen TS. Fast and accurate estimation of


Communicating editor: S. Gravel