AUTOMATIC INFERENCE OF DEMOGRAPHIC PARAMETERS USING GENERATIVE ADVERSARIAL NETWORKS

Zhanpeng Wang[1] **Jiaping Wang**[1] **Michael Kourakos**[2] **Nhung Hoang**[2] **Hyong Hark Lee**[2]

Iain Mathieson[3] **Sara Mathieson**[1,†]

[1] Department of Computer Science, Haverford College, Haverford, PA
[2] Department of Computer Science, Swarthmore College, Swarthmore, PA
[3] Department of Genetics, University of Pennsylvania, Philadelphia, PA
[†] Corresponding author: Sara Mathieson, smathieson@haverford.edu

ABSTRACT

Population genetics relies heavily on simulated data for validation, inference, and intuition. In particular, since real data is always limited, simulated data is crucial for training machine learning methods. Simulation software can accurately model evolutionary processes, but requires many hand-selected input parameters. As a result, simulated data often fails to mirror the properties of real genetic data, which limits the scope of methods that rely on it. In this work, we develop a novel approach to estimating parameters in population genetic models that automatically adapts to data from any population. Our method is based on a generative adversarial network that gradually learns to generate realistic synthetic data. We demonstrate that our method is able to recover input parameters in a simulated isolation-with-migration model. We then apply our method to human data from the 1000 Genomes Project, and show that we can accurately recapitulate the features of real data.

Keywords Evolutionary modeling · Demographic inference · Generative adversarial network · Simulated data

Introduction

Simulation is a key component of population genetics. It helps to train our intuition, and is important for the development, testing, and comparison of inference methods. Because population genetic models such as the ancestral recombination and selection graphs [1, 2] are computationally intractable for inference but relatively easy to simulate, simulations are also heavily used for parameter inference. Approximate Bayesian Computation (ABC) [3] is a widely used example. Regardless of the application, the goal is to simulate data that is “realistic" in the sense that it resembles real data from

the populations of interest. Typically this is done by fixing some parameters that are fairly well-known, for example mutation and recombination rates, and then choosing other parameters to match some property of the real data, usually based on summary statistics. However, this involves searching over a high-dimensional parameter space, and an implicit weighting on the importance of different summary statistics. Often, parameters that create simulations that match one type of summary statistic (e.g. the site frequency spectrum) do not match others (e.g. linkage disequilibrium patterns) [4]. Here, we present a novel parameter learning approach using Generative Adversarial Networks (GANs). In this approach, "realistic" means "cannot be distinguished from real data by a machine learning algorithm", specifically a convolutional neural network (CNN).

Machine learning (ML) methods have been emerging more broadly as promising frameworks for population genetic inference. The high-level goal of training a ML method is to learn a function from the input (genetic data) to the output (evolutionary parameters). Some early efforts used machine learning to account for issues that arise with high-dimensional summary statistics [5–7]. More recently, machine learning approaches have used various forms of convolutional, recurrent, and "deep" neural networks to improve inference [8–12]. One of the goals of moving to these approaches was to enable inference frameworks to operate on the "raw" data (genotype matrices), which avoids the loss of information that comes from reducing genotypes to summary statistics. However, all these algorithms rely heavily on simulated datasets for training. In machine learning more broadly, data is often hand-labeled with "true" values – part of this data is used to train the model, and part is held aside to test the model. In population genetics, training data is extremely limited, and thus all approaches rely on simulations to train and validate ML models.

Current simulators [13–19] are well equipped to replicate mechanisms of evolution, but require many user-selected input parameters including mutation rates, recombination and gene conversion rates, population size changes, natural selection, migration rates, and admixture proportions. We do not always have a good sense of what these parameters should be, especially in understudied populations and non-model species. For example, mutation and recombination rates estimated in one population are frequently used to simulate data for another, despite the fact that these rates differ between human populations [20–24].

Generative models provide one route to simulating more realistic population genetic data. Typically, generative models create artificial data based directly on observed data, without an explicit underlying model. They have been used to create synthetic examples in a wide range of fields, from images and natural language to mutational effects [25] and single cell sequencing [26]. In particular, Generative Adversarial Networks (GANs) work by creating two networks that are trained together [27, 28]. One network (the *generator*) generates simulated data, while the other network (the *discriminator*) attempts to distinguish between "real" data and "fake" (synthetic) simulations. As training progresses,

the generator learns more about the real data and gets better at creating realistic examples, while the discriminator learns to pick up on subtle differences and gets better at distinguishing examples. After training is complete, the generator can be used to create new examples that are indistinguishable (by the discriminator) from real data, but where the ground truth is known (i.e. labeled data).

Here we present a parametric GAN framework. The discriminator is a permutation-invariant CNN that takes as input a genotype matrix and classifies it as real data or synthetic data. Through training, the discriminator tries to get better at this binary classification task. The generator is a coalescent simulator that generates genotype data from a parameterized demographic history. We train the generator using a simulated annealing algorithm that proposes parameter updates leading to more discriminator confusion. We apply our method, called `pg-gan`, in a variety of scenarios to demonstrate that it is able to recapitulate the features of real genetic data. Although we focus on humans, the underlying methodology enables the simulation of any population or species, regardless of how much is known *a priori* about their specific evolutionary parameters. We anticipate that the approach outlined in this work will be useful in evaluating and strengthening the match between simulated and real data, especially for understudied populations that deviate from broad geographic groups. There has also been a push in the population genetics community to standardize simulation resources [29] – we see our method as contributing to the assessment and refinement of published models as they are applied to new datasets.

Our approach is different from that outlined in [30], which uses a GAN to generate artificial genomes that mirror the properties of real genomes. Their approach is a more classical GAN that does not include an evolutionary model, so the resulting artificial genomes are "unlabeled". Such an approach is useful for creating proxy genomes that preserve privacy but still maintain realistic aggregate properties. However this synthetic data could not be used downstream to train or validate supervised machine learning methods since no evolutionary ground truth is known. Our hybrid approach combines the ability of GANs to create realistic data with the interpretability that comes from an explicit model of evolution.

Materials and Methods

At a high level, our method works by simulating data from an underlying evolutionary model, and comparing it to real data via a neural network discriminator. As the discriminator is trained, it tries to minimize a loss function that incentivizes learning the difference between real data and synthetic data. But at the same time, the generator refines the evolutionary model so that it recapitulates the features of real data and attempts confuse the discriminator. At the end, the evolutionary model can be used to simulate additional realistic data for use in downstream applications or method

comparisons. Additionally, the parameters of the final evolutionary model can be interpreted to learn more about the population or species of interest.

This type of GAN framework is not a traditional optimization problem – due to the dual nature of the generator and discriminator there are two optimization problems and the final "GAN confusion" can be used to assess the success of the algorithm – a low confusion (i.e. classification accuracy close to 1) indicates that the simulations are not capturing the real data and the discriminator is easily able to tell the difference between the two types of data. A high confusion (accuracy close to 0.5) ideally indicates the evolutionary model has created simulations that are well-matched to the real data. However, an accuracy close to 0.5 could also mean that the discriminator has not learned anything and is either flipping a coin when classifying examples, or classifying all examples as the same class. See Figure 1 for a diagram of the method. The inputs to the method are an evolutionary model and a set of real data. To create a series of training examples, the real data is divided into regions of length L and the middle biallelic S SNPs from each region are retained (encoded as 0/1). Regions with insufficient SNPs are centered and zero-padded. The simulated data is treated in the same fashion to ensure that the discriminator does not learn the difference between real and simulated data from a pre-processing inconsistency.

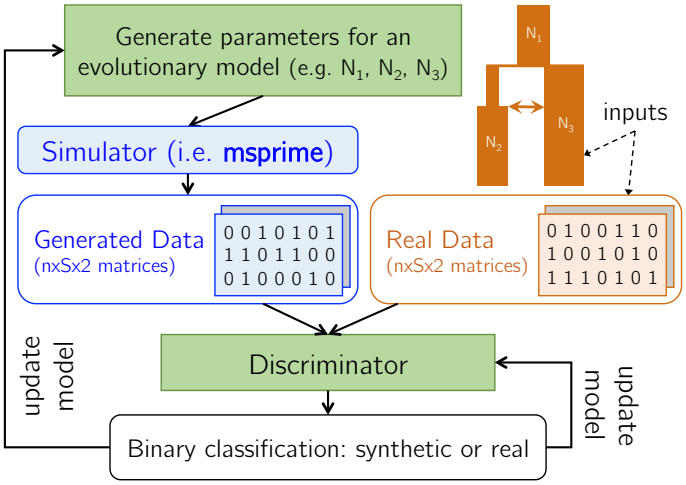

Figure 1: `pg-gan` **algorithm overview.** The inputs to our method are an evolutionary model and a set of real data (orange). The parameters of the generator and discriminator (green) are updated in a unified training framework using simulated annealing (generator) and backpropagation (discriminator). The generated data and real data are analyzed one genotype matrix at a time, where n is the number of haplotypes and S is the number of SNPs retained in each region. Inter-SNP distances are also fed in as a second channel, which provides the discriminator with information about SNP density.

Generator. In image and video generation, the generator often takes the form of a CNN, since a large array of pixel information must be generated from a low-dimensional vector of noise (see Figure 1 of [31] for the architecture of a CNN-based image generator). For our purposes, we do not need to generate the individual genotypes for each training example, but we do need to generate candidate parameters for input into an evolutionary simulator (we use `msprime`

in this study). Let Θ be the set of evolutionary parameters corresponding to model M_Θ. These can be very flexible, including event times, effective population sizes, selection parameters, and rates of mutation, recombination, migration, and exponential growth. The goal of the generator is to select Θ^* that causes the most discriminator confusion, or equivalently, minimizes the test data accuracy. In this binary classification problem, *test accuracy* is defined as the fraction of test datasets where the predicted class matches the true class. We let this test accuracy be denoted $D(\Theta, X_{\text{test}}, y_{\text{test}})$, where X_{test} is held-aside test data (both real examples and examples simulated under M_Θ), and y_{test} is a vector defining the true classes of each example. We occasionally simplify the notation and write the test accuracy as $D(\Theta)$. Using this lens, we can view the generator learning problem as minimizing the multivariate function $D(\Theta)$, while the discriminator is trying to maximize it. We alternate optimizing the discriminator using gradient descent, and optimizing the generator using simulated annealing [32] due to its flexible parameter updates and lack of reliance on an analytic gradient. In simulated annealing, initial parameter values are proposed and then gradually refined. A *temperature* is used to control whether or not new parameter proposals are accepted. The temperature usually begins at a high value, indicating that sub-optimal parameter choices may be accepted liberally to facilitate exploration of the entire parameter space. As training proceeds, the temperature "cools", reducing the chance of accepting a poor parameter choice and allowing the method to converge on a set of parameters that optimizes the desired function.

We initialize each evolutionary parameter by selecting a random value uniformly from a pre-defined range to form $\Theta^{(0)}$. Then the discriminator goes through an initial round of training using simulated data drawn from $M_{\Theta^{(0)}}$. We set the temperature for simulated annealing $T^{(0)} = 1$ and linearly decrease it to 0 over a fixed number of iterations that scales with the number of evolutionary parameters. During each training iteration, several new sets of candidate parameters are proposed, and evaluated based on the resulting discriminator confusion. The new set of parameters is proposed by sampling from a normal distribution around each current value, with variance based on the temperature (this allows the algorithm to explore the parameters space quickly in the beginning, and refine the estimates toward the end of GAN training). More formally, at iteration i, the candidate proposal for parameter p would be

$$\Theta_p^{(\text{proposal})} \sim \mathcal{N}\left(\Theta_p^{(i)}, \sigma^2 \cdot T^{(i)}\right)$$

where σ^2 is the initial variance, which is based on the range of plausible values for each parameter. Out of the several candidate proposals, we choose the one that minimizes $D(\Theta, X_{\text{test}}, y_{\text{test}})$. Then we compare the test accuracy of the chosen proposal to that of the previous iteration. If the proposal reduces or maintains the test accuracy, we always accept it (down to a floor of 0.5, so we do not incentivize flipped classification results). If not, we use the simulated annealing temperature to help define a threshold for acceptance. Formally, if the proposal is Θ and the current parameter

value at iteration i is $\Theta^{(i)}$, then the acceptance probability is

$$p_{\text{accept}} = \frac{1 - |2D(\Theta) - 1|}{1 - |2D(\Theta^{(i)}) - 1|} \cdot T^{(i)}.$$

If we accept the proposed parameters then we set $\Theta^{(i+1)} \leftarrow \Theta$ and run a round of discriminator training using these parameters. An important point is that we do not train the discriminator using the new parameter proposals unless they are accepted. During the candidate proposal phase, we are evaluating the parameter choices by *testing* only.

Discriminator. For the architecture of the discriminator, we use a permutation-invariant CNN based on `defiNETti` [8]. Each example within X_{train} and X_{test} has shape $(n, S, 2)$ where n is the number of haplotypes in the sample, S is the number of retained SNPs in a region of size L, and 2 indicates there is one channel for the genotypes and one channel for inter-SNP distances. The inter-SNP distances are duplicated down each column to allow this slice of the tensor to have the same shape as the genotype information. This also ensures that each convolutional filter processes the genotypes and associated distances at the same time. Alternatively, the convolutional layers can be used on the genotypes only, and the distances concatenated later as a vector. However, this approach does not allow the processing of the two channels to be as tightly coupled. We use convolutional filters of shape 1×5 (1 haplotype, 5 SNPs) to ensure that the order of haplotypes does not impact the results. After several convolutional layers, we condense the output by applying a column-wise permutation-invariant function. We experiment with both `max` and `sum` as permutation-invariant functions, and discuss the advantages of each. For models that consider multiple populations, we augment this framework to include separate permutation-invariant components for each population, then concatenate the flattened output before input into the dense layers at the end of the network. An illustration of our discriminator architecture is shown in Figure 2.

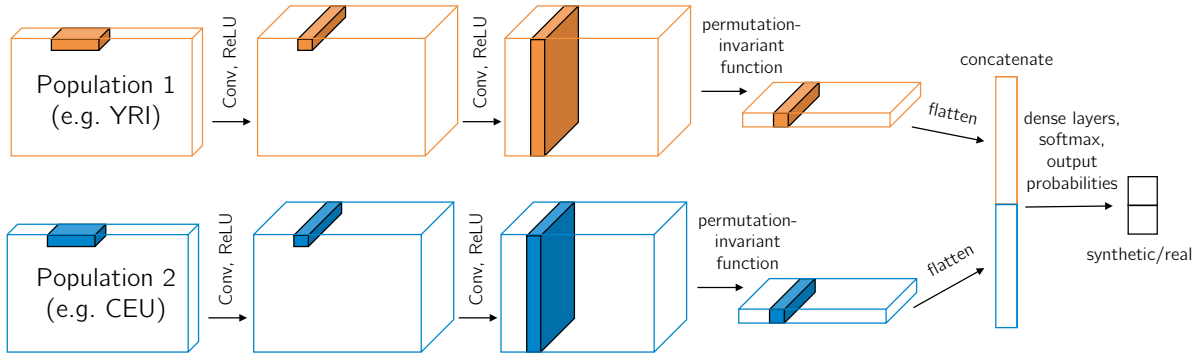

Figure 2: **Multi-population discriminator architecture.** Each example region is of shape $(n, S, 2)$ where n is the number of haplotypes (usually with $n/2$ from population 1 and $n/2$ from population 2). Note that the convolutional filters for population 1 and 2 are shared (i.e. not separate weights) so that haplotype commonalities can be more easily identified.

We train the discriminator using mini-batches of 50 training examples (chosen randomly such that roughly half are real and half are simulated). For each training iteration, we perform 200 batch training updates if the proposed parameters are accepted. This allows the discriminator to learn gradually, as the parameters are being refined. While a test accuracy close to 0.5 is desired by the end of training, the discriminator test accuracy may be close to this value early on in the training process simply because it has not learned anything yet. The goal is for the discriminator to be optimized to distinguish real from simulated data as much as possible and *still* be wrong half the time.

Simulation study. To validate our approach, we first select the "real" dataset to be a simulated one, so that we can test whether the inferred parameters are correct. To assess a variety of different types of parameters, we choose an isolation-with-migration model (see Figure 3A) with six parameters. The parameters include three effective population sizes (N_{anc} for the ancestral population size, and N_1 and N_2 for the sizes of each population after the split). We also infer the split time T_{split}, and the strength of a directional migration pulse at time $T_{\text{split}}/2$. Finally, we infer the per-base, per-generation recombination rate. We evaluate the inferred parameters based on how well they match the "real" parameters.

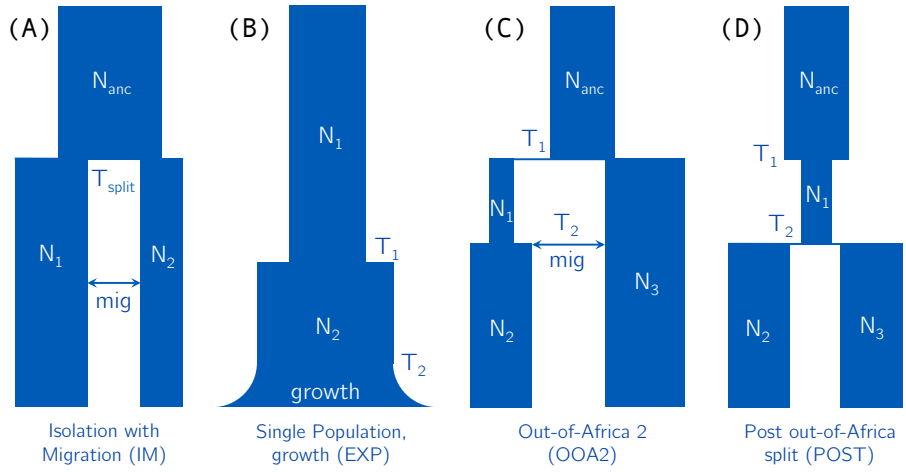

Figure 3: **Set of models.** (A) A six parameter, two population isolation-with-migration model, which we use in the simulation study. The migration event is a single pulse at time $T_{\text{split}}/2$, and can be in either direction. The final parameter (not shown in this diagram) is the recombination rate. (B) A five parameter, single population exponential growth model, which we use to infer histories for YRI, CEU, and CHB separately. (C) A seven parameter, two population model, which we fit separately for YRI/CHB and YRI/CEU. The migration can be in either direction. (D) A six parameter, two population model which we fit to CEU/CHB.

1000 Genomes data analysis. To demonstrate the effectiveness of our method on real data, we use the method to infer demographic parameters for both single- and multi-population models in humans. To ensure that the real data is as similar as possible to the simulated data, we run several pre-processing steps. We first divide each chromosome into $L = 50$kb regions. For each region, we retain it if at least 50% of the bases are inside callable regions, as defined by the "20120824" strict mask [33]. We filter out non-segregating and multi-allelic sites. For both the real and simulated

data we recode the genotypes by setting the minor allele to the value "1" and the major allele to the value "0" so that the discriminator cannot learn to distinguish real data based on reference bias or ancestrally misidentified states. We select the test data randomly and hold it aside so it is not used for training. To mitigate the effects of correlated nearby regions, local variations in mutation and recombination rate, we sample regions randomly (for both training and testing). To avoid processing the real data on-the-fly during training, we follow a data extraction pipeline to convert the real regions into HDF5 format [34, 35]. From each 50kb region we retain a fixed number of SNPs (either 24 or 36) and the associated inter-SNP distances. The number of haplotypes is flexible (due to the permutation-invariant framework), so we use between 196 and 198, matching the minimum number of individuals in each 1000 Genomes population.

For the single population analysis, this leaves 49,276 training examples (91.5% of all regions). For the two-population models, we keep the same total sample size, taking half the haplotypes from each population. This enables us to take two training examples from each region for a total of 98,552 examples for the two-population models.

We test three models (Figure 3B-D). The single-population model has five parameters: two effective population sizes N_1 and N_2, two size-change points T_1 and T_2, and the rate of exponential growth in the recent past. We fit this model to three human populations from the 1000 Genomes project: YRI (West African), CEU (European), and CHB (East Asian). The second model is a simplified two-population Out-of-Africa model. There are seven parameters: four effective population sizes, two time-change points, and a migration pulse that can be in either direction, allowing for migration between African and non-African populations. We fit this model to two pairs of populations: YRI/CEU and YRI/CHB. The third model represents the post-out-of-Africa split between the ancestors of Europeans and East Asians. In this six parameter model we do not include migration, but allow a pre-split bottleneck. We fit this model to the pair of populations CEU/CHB.

Summary statistics. As a qualitative assessment of our results, we compare summary statistics computed on both the real data and data simulated under our inferred parameters. We note that our goal is explicitly *not* to match summary statistics, as matching some types of statistics can bias the resulting fitted model. In addition, we currently do not have an exhaustive or sufficient set of summary statistics that could be used to identify model parameters directly in a likelihood framework. However, summary statistics can give us a sense of which features of real data agree with our simulations and which do not.

To that end, we use seven summary statistics. In all cases, we use 5000 regions of real data and 5000 regions of simulated data to compute the statistics. Each region is 50kb long, with 36 SNPs retained.

- **SFS**: we compute the site frequency spectrum (SFS) by counting the number of singletons, doubletons, etc in each of 5000 regions of real and simulated data. We plot the first 10 entries.

- **Inter-SNP distances**: we plot the distribution of inter-SNP distances for both the real and simulated data (measured in base pairs). This provides a general measure of SNP density.
- **LD**: we compute linkage-disequilibrium (LD) by clustering pairs of SNPs based on their inter-SNP distance. We divide these distances into 10 bins and average the correlation r^2 within each one.
- **Tajima's D**: we plot the distribution of Tajima's D, computed separately for each region.
- **Pairwise heterozygosity**: we plot the distribution of pairwise heterozygosity (π), computed separately for each region.
- **Number of haplotypes**: we plot the distribution of number of haplotypes for each region.
- $F_{\mathbf{st}}$: for the two-population split models, we use Hudson's F_{st} to measure population differentiation [36].

Our `pg-gan` software uses a `tensorflow` [37] backend and is available open-source at `https://github.com/mathiesonlab/pg-gan`. Data Availability Statement: all data included in this work is publicly available through the 1000 Genomes Project `https://www.internationalgenome.org/` [33].

Results

Simulation study. To validate our method, we first create a "real" dataset using simulated data, so we know the true evolutionary parameters. Using the six-parameter IM model described in the Methods section, we begin by inferring one parameter at a time, fixing all others to their true values. We initialize the parameter to infer by choosing a random value uniformly from the parameter's range, defined in Table 1. Throughout, we also fix the mutation rate to 1.25×10^{-8} per base per generation.

Table 1: **Parameter ranges.** When inferring a parameter, we initialize its value by drawing a value uniformly from the given ranges. For each parameter update, we do not allow the parameter to go outside its range. Whenever we noticed a parameter moving against a boundary, we increased the range. Overall the ranges are meant to be plausible values based on previous studies or reasonable evolutionary events.

Parameter	min	max	units
N_A	1000	20000	individuals
N_B	1000	20000	individuals
Ne	1000	20000	individuals
reco	1×10^{-9}	1×10^{-7}	per base per generation
N_{anc}	1000	25000	individuals
T_{split}	500	20000	generations
mig	-0.2	0.2	per generation
N_1	1000	30000	individuals
N_2	1000	30000	individuals
growth	0	0.05	per generation
N_3	1000	30000	individuals
T_1	1500	5000	generations
T_2	100	1500	generations

For the parameter updates, we set the initial variance σ^2 to the parameter range divided by 10. During each iteration, we choose 5 independent candidate parameters, and select the one that minimizes test accuracy. We tested updating one parameter each iteration vs. updating all the parameters each iteration, and generally found that updating one at a time led to more stable and consistent results.

We ran our method on the full set of six parameters for this model, performing joint inference. The results are shown in Figure 4. Points closer to red indicate earlier in the GAN training, and blue indicate the end of training. The y-axis shows the test accuracy. We see that in general, `pg-gan` is able to find parameter values that cause the optimal discriminator test accuracy of 0.5 (the final test accuracy here was 0.518), and the inferred values are close to the true values used for the "real" data. Some of the training iteration points are aligned vertically since we update one parameter at a time – if a parameter is not updated that iteration, the test accuracy could change while that parameter value remains the same. Note that sometimes early in training, the test accuracy is low (close to 0.5) simply because the discriminator has learned nothing and essentially guesses randomly.

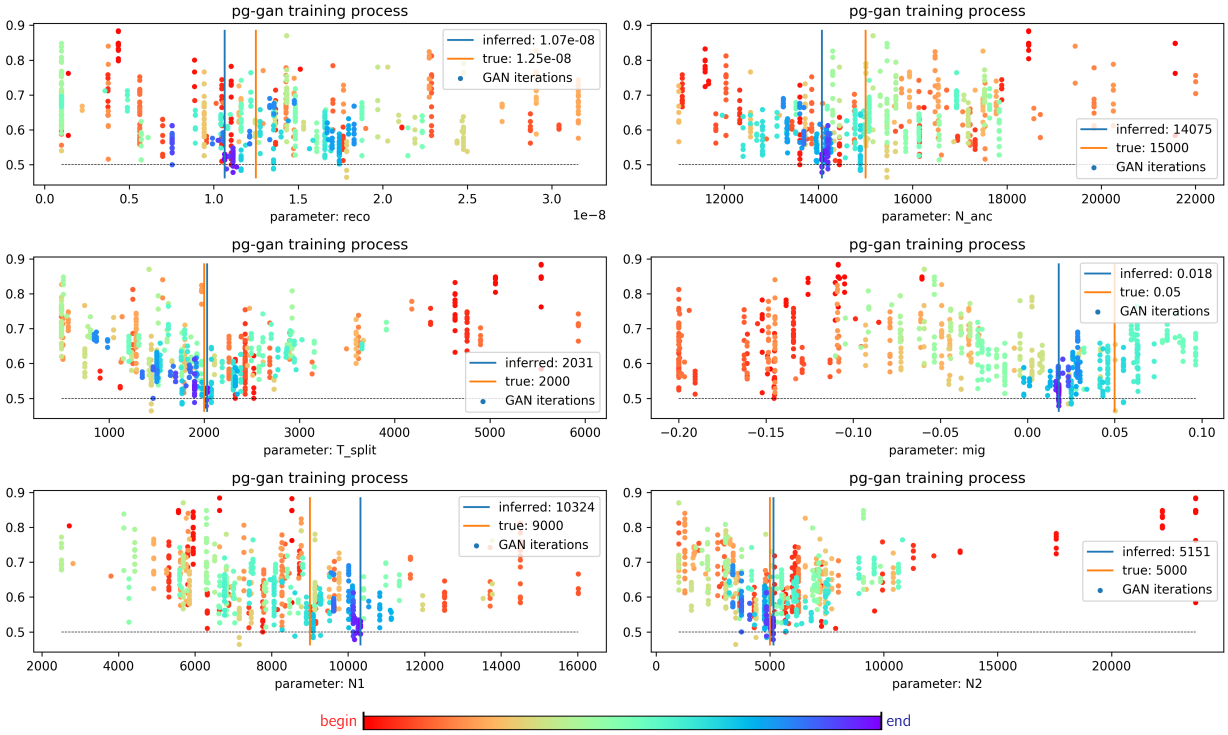

Figure 4: **IM model joint parameter inference on simulated data.** In this scenario we jointly infer this six parameters of the IM model from Figure 3A. Each point shows the discriminator accuracy and parameter value at a single iteration. Colors indicate the training iteration, with red being closer to the beginning of training and blue closer to the end.

1000 Genomes data analysis. We first analyzed three populations (YRI, CEU, and CHB) separately, each under the five-parameter model with recent exponential growth. We pre-processed the data and modified the simulations in a

variety of ways to test the effect on the discriminator accuracy. For all results presented below, we used $n = 198$ (size of CEU) and $S = 36$. We group our experiments into algorithmic changes and modeling changes. In terms of algorithmic modifications (see Figure 5A), we use `max` as the permutation-invariant function in the CNN architecture, and then `sum`. We generally find that `sum` causes the discriminator to learn more slowly, allowing the generator time to find good parameter choices. `max` sometimes causes the discriminator to converge quickly, easily distinguishing the real from simulated data before the generator can move to a promising location in the parameter space. We also experiment with updating one parameter at a time, rather than all parameters. For CEU in particular, algorithmic changes did not change the results dramatically, but for YRI, using the `sum` function and updating one parameter at a time allowed the training process to find more realistic parameters.

In terms of modeling changes (Figure 5B), we first fit a one-parameter demographic model with a single constant population size, fixing both the recombination and mutation rates to 1.25×10^{-8} per base per generation. We contrast this result with the five-parameter exponential growth model. Finally, we allow the recombination rate to vary by drawing from the distribution of HapMap combined recombination rates [38]. Both these modifications allow for more flexible simulations, which improved our results. The summary of these results for all populations is shown in Figure 5. The minimum discriminator accuracy is around 60%. That is, the discriminator can still distinguish real and simulated data slightly better than random, probably indicating that there are still features of the data that we are not capturing.

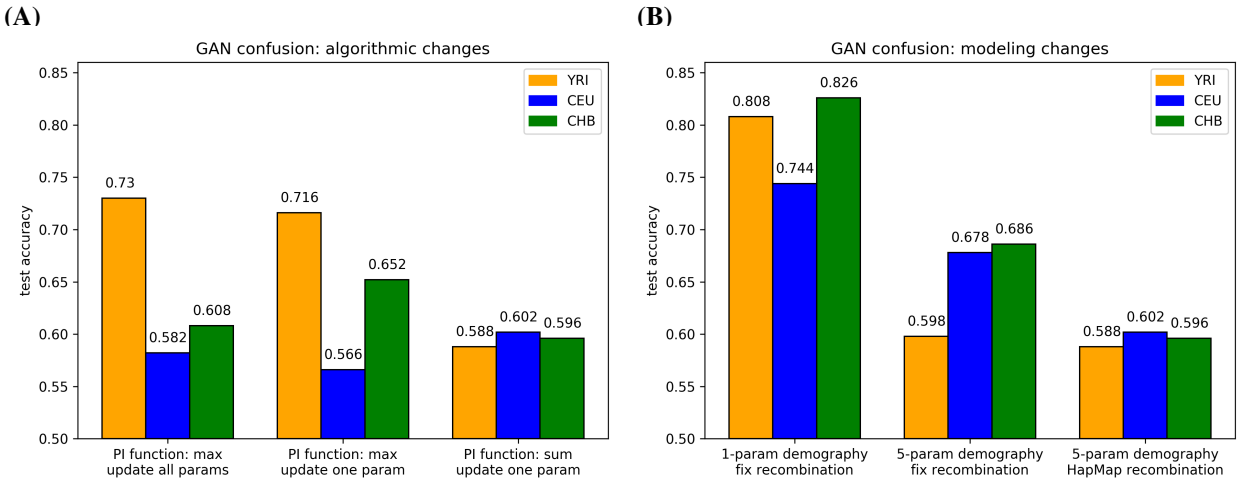

Figure 5: **1000 Genomes single population analysis.** (A) Results of algorithmic modifications. We vary the permutation-invariant (PI) function between `max` and `sum`, with `sum` usually producing better results. We also vary the parameter update approach (all at once vs. one at a time). (B) Results of modeling modifications. We use a constant population size for the first group of bars, then move to the five-parameter exponential growth model (Figure 3B). We also sample recombination rates from HapMap in the last group of bars, instead of fixing the recombination rate. Note that the last set of bars in both figures is the same, representing our optimal algorithmic and modeling choices.

Inferred parameters for each population under the five-parameter exponential growth model are shown in Table 2, and correspond to the optimal algorithm and modeling choices from Figure 5. The Out-of-Africa bottleneck (N_2) is very

apparent in CEU and CHB, with a much more modest reduction in YRI. The per-generation growth rate for YRI is likely overestimated, possibly because the start of exponential growth (T_2) is underestimated.

Table 2: **1000 Genomes single population parameter inference.** Inferred parameters for the exponential growth model (see Figure 3B) in YRI, CEU, and CHB. These parameters correspond to the optimal algorithm and modeling choices from the last set of bars in Figure 5. We generally infer similar parameters for CEU and CHB.

Population	N_1	N_2	growth	T_1	T_2
YRI	29,781	18,404	0.0498	1,989	309
CEU	24,121	5,448	0.0104	3,287	526
CHB	23,055	5,079	0.0103	3,837	677

As an independent assessment of our results, we compare the summary statistics between real data and data simulated under the parameter choices corresponding to various scenarios from Figure 5 (see Methods for a description of the summary statistics). In Figure 6 we show two sets of summary statistics for YRI. On the left we show the five-parameter demography results with a fixed recombination rate, and on the right we use the HapMap recombination rates. While some statistics match closely, others are less well-matched, consistent with the discriminator being imperfectly confused. Full sets of summary statistics for YRI, CEU, and CHB are shown in Figures S1-S3.

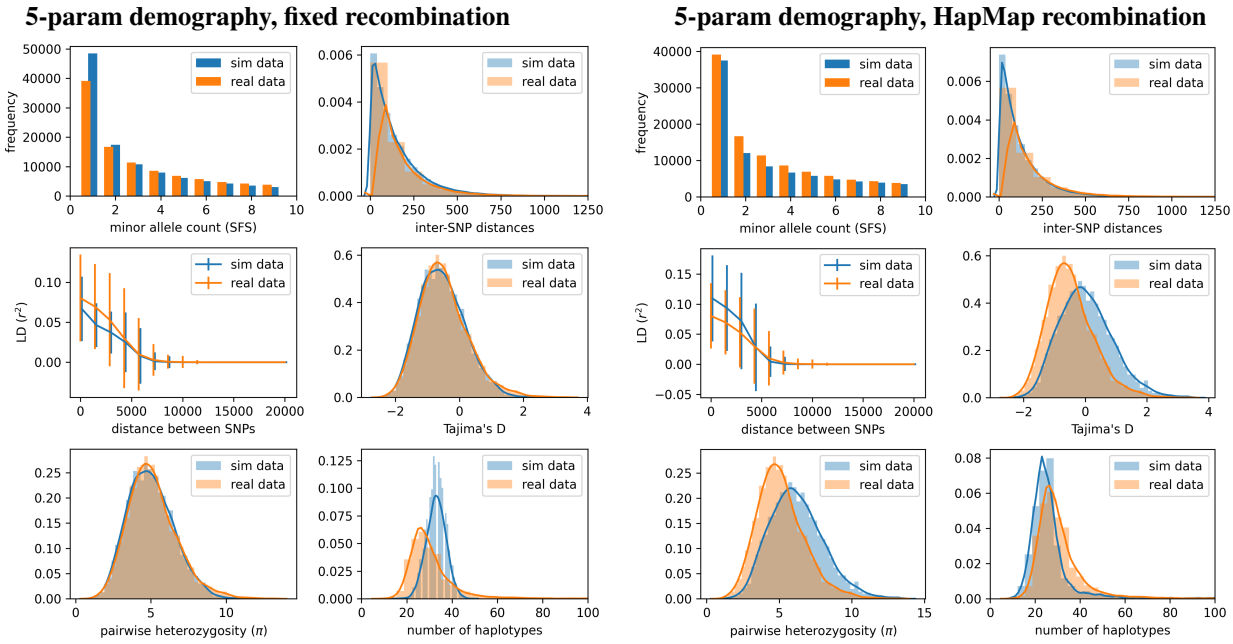

Figure 6: **YRI: single population model.** Summary statistic comparison between YRI data from the 1000 Genomes Project and data simulated under our inferred parameters for each scenario. Left: simulated data under the 5-parameter exponential growth model with a fixed recombination rate. Right: same exponential growth model but with recombination rates sampled from the HapMap recombination map.

For all our real data results, we use two checks to ensure that a low test accuracy is not simply the result of the discriminator failing to learn anything. First we look for a high test accuracy at some point during training, which

typically means that for some parameter combinations and discriminator weights, the real and simulated data were easily distinguished. Second, we check the final confusion matrix for the test data to make sure that all datasets are not simply classified in the same way (all real or all synthetic). Both these checks are satisfied for all real data analysis.

Lastly, we ran our method on 1000 Genomes data from two populations. To represent the Out-of-Africa event, we use two pairs of populations separately: YRI/CEU and YRI/CHB, using the model from Figure 3C. We also use CEU/CHB with the model from Figure 3D to represent the post-out-of-Africa split between the ancestors of Europeans and East Asians. The resulting test accuracies are shown in Figure 7. The YRI/CEU and YRI/CHB results are comparable with the single population analysis, but the CEU/CHB test accuracy is much higher, indicating that this model or the resulting parameter inference is not a good match for this dataset. For YRI/CEU and YRI/CHB, we provide the parameter inference results in Table 3. For both pairs of populations, we infer the "back-migration" of ancestors of non-Africans back into Africa observed in [39]. Here, this is represented as a single migration pulse with a negative migration proportion.

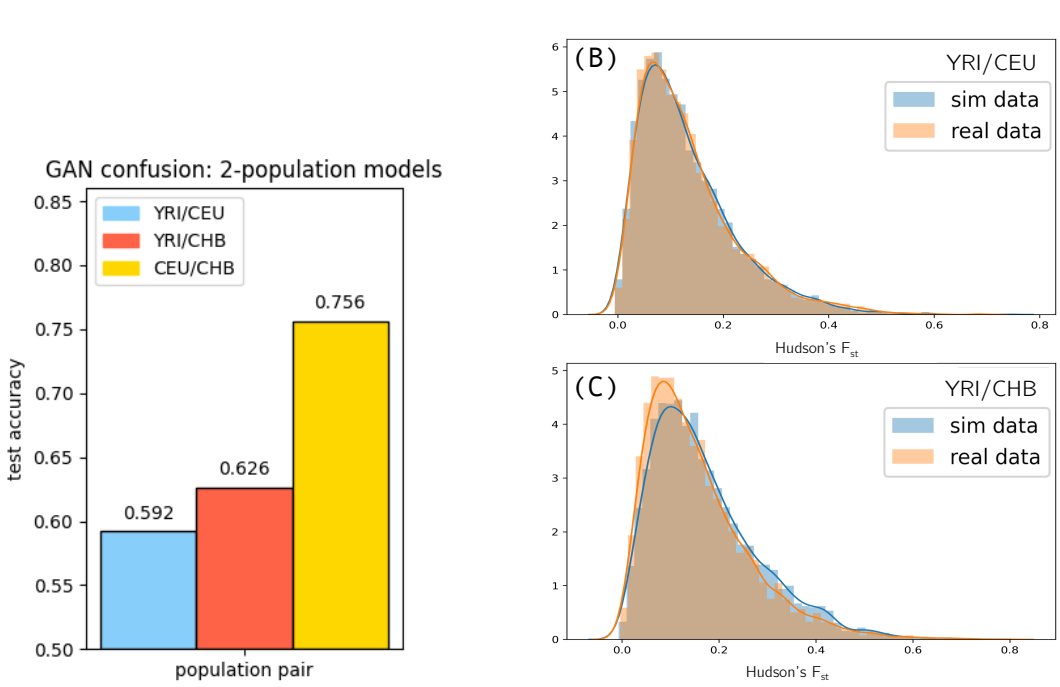

Figure 7: **1000 Genomes two population analysis.** (A) Test accuracy results on the population split models for YRI/CEU, YRI/CHB, and CEU/CHB. The Out-of-Africa models and parameter inference for YRI/CEU and YRI/CHB generally seem to do well, but the CEU/CHB split model and/or parameter inference does not result in simulated data that matches real data (much more easily distinguishable). (B-C) F_{st} summary statistic comparison for the YRI/CEU and YRI/CHB splits.

In the two-population scenario, we compute summary statistics on each population separately, shown in Figure 8 for the YRI/CEU split. We see close agreement between the statistics for real and simulated data in the two-population Out-of-Africa scenario, indicating that modeling the additional complexity of the split leads to more realistic simulated

Table 3: **1000 Genomes two-population parameter inference.** Inferred parameters for the Out-of-Africa model (see Figure 3C) fit to YRI/CEU and YRI/CHB. We generally see similar results for both pairs of populations, with a lower test accuracy for YRI/CEU, indicating a closer match to the real data.

Populations	N_{anc}	mig	N_1	N_2	N_3	T_1	T_2
YRI/CEU	21,763	-0.0796	4,263	26,975	29,803	3,647	1,051
YRI/CHB	24,782	-0.1117	3,924	28,176	29,302	4,919	1,499

data. YRI/CHB statistics are shown in Figure S4 – for the YRI samples these statistics are not quite as closely matched, which fits with the slightly higher test accuracy for this scenario. CEU/CHB statistics are shown in Figure S5 – in this scenario the simulated statistics show deviation from the real data, also fitting with the higher test accuracy.

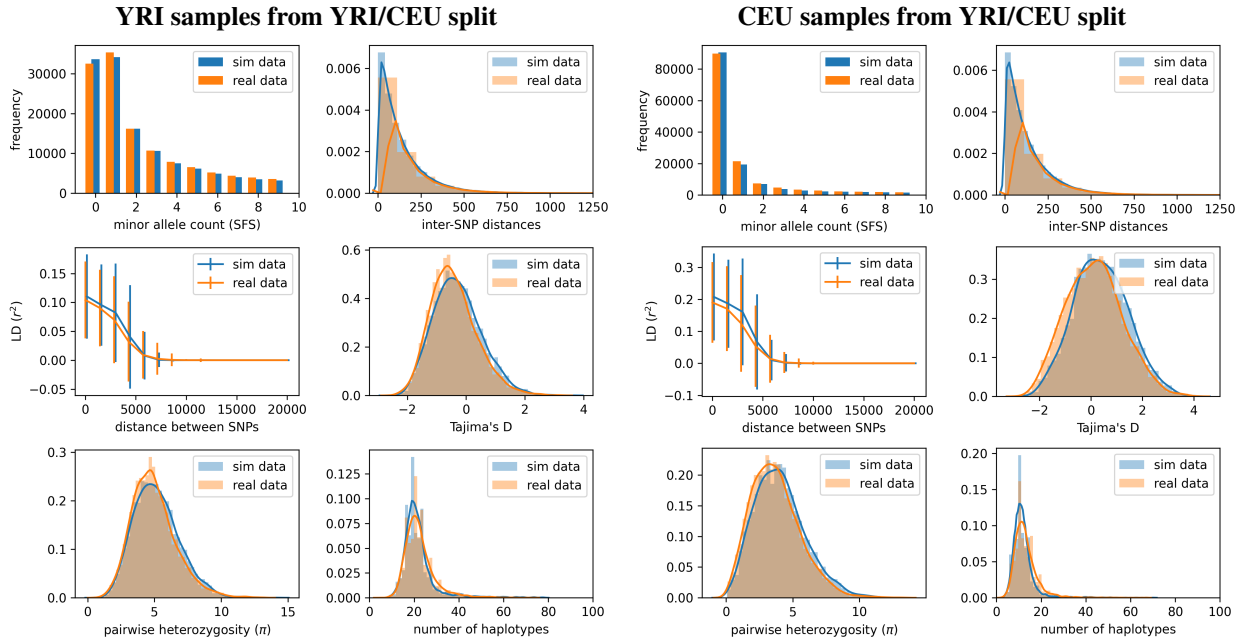

Figure 8: **YRI/CEU: two population model.** Summary statistic comparison real 1000 Genomes data and data simulated under the inferred parameters from Table 3 (first row). Left: statistics computed on YRI samples only. Right: statistics computed on CEU samples only. Sites with count zero are segregating in only one population.

Computational Resources. The runtime of our method is around 12-15 hours using a Quadro P5000 GPU. As the number of parameters increases, we increase the number of training iterations linearly, which adds to the runtime. Pre-processing the real data takes 8-10 hours per population or scenario (i.e. two population split model). The resulting file sizes are 5.3G for single populations and 11G for pairs of populations, which are loaded into memory during training. To reduce the demand for RAM, it would also be possible to break up these files into smaller units and to load them in as needed during training.

Discussion

In this work we present a method for automatically creating realistic simulated genetic data. Our `pg-gan` algorithm is a more holistic approach to parameter inference than methods that are based on summary statistics. Our generative adversarial framework simultaneously trains a generator to produce reasonable evolutionary parameters and a discriminator to distinguish real data from simulated. We use real data during training to make sure the simulations capture realistic genomic features. We demonstrate the use of our method in an isolation-with-migration simulation setting and create simulated data that mirrors three human populations individually, and in pairs. In these single-population models, we achieve discriminator accuracy close to 60%, indicating strong, albeit imperfect, confusion between the real and simulated data. The approach is highly flexible and can automatically fit any parameterized model to any genomic data. It is particularly useful for understudied populations or species, since any unknown parameters can be included in the model and learned.

The approach also yields a natural way of evaluating and refining simulation pipelines. If simulations are easily distinguished from real data, then the model is not producing realistic simulations. We easily reach perfect (50%) discriminator confusion in simulations but with real data, even for humans where we have a good understanding of parameter ranges and likely models, our best accuracy is around 60%. This is likely because there are features of the real data that our models do not include, for example heterogeneity in mutation rates, limited power to detect rare variants, inaccessible regions of the genome, and the effects of natural selection. It would be possible to incorporate these effects and evaluate their impact. In particular, heterogeneity can be modeled by fitting a distribution to parameters, rather than a point estimate. More generally, in our current implementation, the topology of the demographic model needs to be specified ahead of time. However, it would be possible to extend our method to explore the space of demographic models more broadly to automatically learn the topology as well as the parameters of the model. In that case, many different models would likely produce data indistinguishable from real data.

A concern for training the discriminator is overfitting due to unbalanced training data. The amount of real data is fixed, but the number of simulated examples can be unlimited. There are many ways to guard against overfitting neural networks, including regularization, dropout [40], and architecture modifications. An important line of future research will include optimizing the training procedure in the presence of limited real data. In addition to discriminator choices, the real data can also be processed to maximize the number of regions, including reducing the number of haplotypes (which allows for multiple regions per locus), reducing the size of each locus, or increasing the number of SNPs retained from each locus (retaining 36 SNPs from a 50kb region is generally very conservative in humans, and it is not a problem if some regions have fewer SNPs). This type of optimization needs to be balanced with potential loss of information.

Another approach is to make use of transfer learning [41]. In transfer learning, the parameters of an ML model are initialized by training on a large dataset, then "fine-tuned" by training on a smaller number of examples from the target dataset. In our case, a large dataset like the 1000 Genomes could be used to find a "good guess" for weights of the discriminator, then these parameters could be fine-tuned using data with fewer regions or sequenced individuals.

Another potential asymmetry comes from the fact that the generator parameters are being updated at the same time that the discriminator is being trained. The training of both components needs to be balanced – if the discriminator learns the difference between real and simulated data too quickly, the generator might not have a chance to explore a parameter space that would actually cause confusion. On the other hand, if the discriminator learns too slowly, all generator updates might look equally confusing. In our experiments, the former situation is more common, but the path of training should always be monitored to guard against the latter.

Future developments will include integrating more realistic features of real data and applying our approach to non-human species. In terms of methodological development, we aim to integrate transfer learning and develop interpretative approaches for the CNN discriminator, in order to investigate alignment between its hidden layers and traditional summary statistics. Modern machine learning has proved to be powerful in many domains, and our work emphasizes that this is true for population genetics as well. However, machine learning in population genetics requires novel architectures, for example our parametric generator and multi-population CNN discriminator – innovations that will be useful for future development of ML methods in the field.

Acknowledgments

The authors would like to thank Joe Cammisa for extensive computational support.

References

[1] Robert C Griffiths and Paul Marjoram. An ancestral recombination graph. In *IMA*, volume 87, page 257, 1997.

[2] Claudia Neuhauser and SM Krone. Ancestral processes with selection. *Theor. Pop. Biol*, 51:210–237, 1997.

[3] Mark A Beaumont, Wenyang Zhang, and David J Balding. Approximate Bayesian computation in population genetics. *Genetics*, 162(4):2025–2035, 2002.

[4] Annabel C Beichman, Tanya N Phung, and Kirk E Lohmueller. Comparison of single genome and allele frequency data reveals discordant demographic histories. *G3: Genes, Genomes, Genetics*, 7(11):3605–3620, 2017.

[5] Michael GB Blum and Olivier François. Non-linear regression models for Approximate Bayesian Computation. *Statistics and computing*, 20(1):63–73, 2010.

[6] Roy Ronen, Nitin Udpa, Eran Halperin, and Vineet Bafna. Learning natural selection from the site frequency spectrum. *Genetics*, 195(1):181–193, 2013.

[7] Sara Sheehan and Yun S Song. Deep learning for population genetic inference. *PLoS computational biology*, 12 (3):e1004845, 2016.

[8] Jeffrey Chan, Valerio Perrone, Jeffrey Spence, Paul Jenkins, Sara Mathieson, and Yun Song. A likelihood-free inference framework for population genetic data using exchangeable neural networks. In *Advances in Neural Information Processing Systems*, pages 8594–8605, 2018.

[9] Lex Flagel, Yaniv Brandvain, and Daniel R Schrider. The unreasonable effectiveness of convolutional neural networks in population genetic inference. *Molecular biology and evolution*, 36(2):220–238, 2019.

[10] Luis Torada, Lucrezia Lorenzon, Alice Beddis, Ulas Isildak, Linda Pattini, Sara Mathieson, and Matteo Fumagalli. ImaGene: a convolutional neural network to quantify natural selection from genomic data. *BMC bioinformatics*, 20(9):337, 2019.

[11] Jeffrey R Adrion, Jared G Galloway, and Andrew D Kern. Predicting the landscape of recombination using deep learning. *Molecular Biology and Evolution*, 37(6):1790–1808, 2020.

[12] Théophile Sanchez, Jean Cury, Guillaume Charpiat, and Flora Jay. Deep learning for population size history inference: design, comparison and combination with approximate Bayesian computation. *BioRxiv*, 2020.

[13] Richard R Hudson. Generating samples under a Wright–Fisher neutral model of genetic variation. *Bioinformatics*, 18(2):337–338, 2002.

[14] Kosuke M Teshima and Hideki Innan. mbs: modifying Hudson's ms software to generate samples of DNA sequences with a biallelic site under selection. *BMC bioinformatics*, 10(1):166, 2009.

[15] Gregory Ewing and Joachim Hermisson. MSMS: a coalescent simulation program including recombination, demographic structure and selection at a single locus. *Bioinformatics*, 26(16):2064–2065, 2010.

[16] Laurent Excoffier, Isabelle Dupanloup, Emilia Huerta-Sánchez, Vitor C Sousa, and Matthieu Foll. Robust demographic inference from genomic and SNP data. *PLoS Genet*, 9(10):e1003905, 2013.

[17] Andrew D Kern and Daniel R Schrider. Discoal: flexible coalescent simulations with selection. *Bioinformatics*, 32(24):3839–3841, 2016.

[18] Jerome Kelleher, Alison M Etheridge, and Gilean McVean. Efficient coalescent simulation and genealogical analysis for large sample sizes. *PLoS computational biology*, 12(5):e1004842, 2016.

[19] Benjamin C Haller and Philipp W Messer. SLiM 3: Forward genetic simulations beyond the Wright–Fisher model. *Molecular biology and evolution*, 36(3):632–637, 2019.

[20] Anjali G Hinch, Arti Tandon, Nick Patterson, Yunli Song, Nadin Rohland, Cameron D Palmer, Gary K Chen, Kai Wang, Sarah G Buxbaum, Ermeg L Akylbekova, et al. The landscape of recombination in African Americans. *Nature*, 476(7359):170–175, 2011.

[21] Kelley Harris. Evidence for recent, population-specific evolution of the human mutation rate. *Proceedings of the National Academy of Sciences*, 112(11):3439–3444, 2015.

[22] Kelley Harris and Jonathan K Pritchard. Rapid evolution of the human mutation spectrum. *Elife*, 6:e24284, 2017.

[23] Iain Mathieson and David Reich. Differences in the rare variant spectrum among human populations. *PLoS genetics*, 13(2):e1006581, 2017.

[24] Michael D Kessler, Douglas P Loesch, James A Perry, Nancy L Heard-Costa, Daniel Taliun, Brian E Cade, Heming Wang, Michelle Daya, John Ziniti, Soma Datta, et al. De novo mutations across 1,465 diverse genomes reveal mutational insights and reductions in the Amish founder population. *Proceedings of the National Academy of Sciences*, 117(5):2560–2569, 2020.

[25] Adam J Riesselman, John B Ingraham, and Debora S Marks. Deep generative models of genetic variation capture the effects of mutations. *Nature methods*, 15(10):816–822, 2018.

[26] Romain Lopez, Jeffrey Regier, Michael B Cole, Michael I Jordan, and Nir Yosef. Deep generative modeling for single-cell transcriptomics. *Nature methods*, 15(12):1053–1058, 2018.

[27] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[28] Mehdi Mirza and Simon Osindero. Conditional generative adversarial nets. *arXiv preprint arXiv:1411.1784*, 2014.

[29] Jeffrey R Adrion, Christopher B Cole, Noah Dukler, Jared G Galloway, Ariella L Gladstein, Graham Gower, Christopher C Kyriazis, Aaron P Ragsdale, Georgia Tsambos, Franz Baumdicker, et al. A community-maintained standard library of population genetic models. *BioRxiv*, pages 2019–12, 2020.

[30] Burak Yelmen, Aurélien Decelle, Linda Ongaro, Davide Marnetto, Corentin Tallec, Francesco Montinaro, Cyril Furtlehner, Luca Pagani, and Flora Jay. Creating artificial human genomes using generative models. *bioRxiv*, page 769091, 2019.

[31] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.

[32] Martin Pincus. Letter to the editor—a Monte Carlo method for the approximate solution of certain types of constrained optimization problems. *Operations research*, 18(6):1225–1228, 1970.

[33] 1000 Genomes Project Consortium et al. A global reference for human genetic variation. *Nature*, 526(7571): 68–74, 2015.

[34] Alistair Miles. *Extracting data from VCF files*, 2017. URL `http://alimanfoo.github.io/2017/06/14/read-vcf.html`.

[35] Alistair Miles. *Estimating Fst*, 2015. URL `http://alimanfoo.github.io/2015/09/21/estimating-fst.html`.

[36] Richard R Hudson, Montgomery Slatkin, and Wayne P Maddison. Estimation of levels of gene flow from DNA sequence data. *Genetics*, 132(2):583–589, 1992.

[37] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL `https://www.tensorflow.org/`. Software available from tensorflow.org.

[38] International HapMap Consortium et al. A second generation human haplotype map of over 3.1 million SNPs. *Nature*, 449(7164):851, 2007.

[39] Christopher Bernard Cole, Sha Joe Zhu, Iain Mathieson, Kay Prfüer, and Gerton Lunter. Ancient admixture into Africa from the ancestors of non-Africans. *bioRxiv*, 2020.

[40] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

[41] Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 22(10):1345–1359, 2010.

Supplementary Material

Automatic inference of demographic parameters using Generative Adversarial Networks

Zhanpeng Wang[1], Jiaping Wang[1], Michael Kourakos[2], Nhung Hoang[2], Hyong Hark Lee[2], Iain Mathieson[3], Sara Mathieson[1,†]

[1] Department of Computer Science, Haverford College, Haverford, PA
[2] Department of Computer Science, Swarthmore College, Swarthmore, PA
[3] Department of Genetics, University of Pennsylvania, Philadelphia, PA
[†] Corresponding author: Sara Mathieson, smathieson@haverford.edu

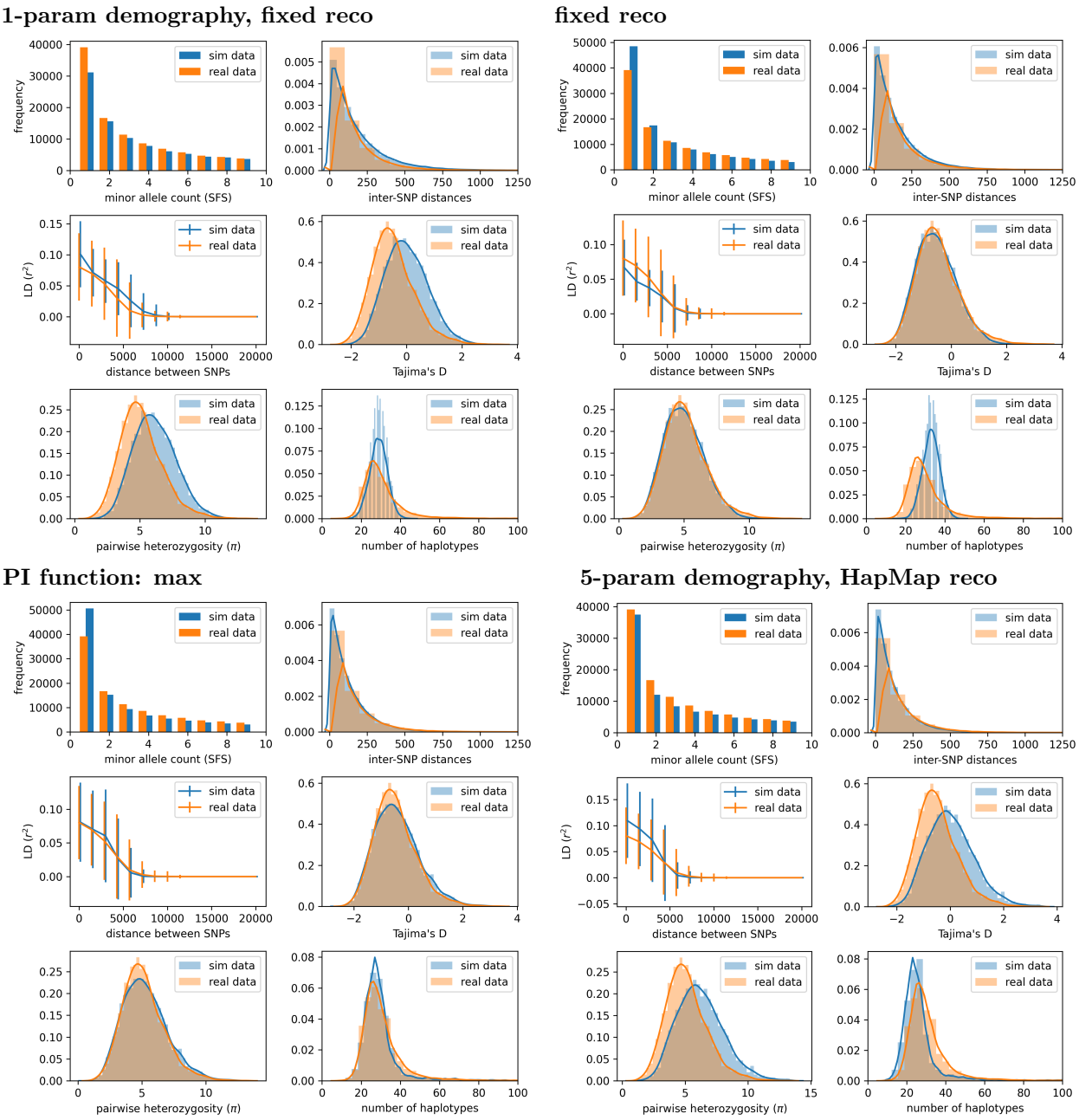

Figure S1: **YRI: single population model.** Summary statistic comparison between YRI data from the 1000 Genomes project and simulated data under the inferred parameters from the various scenarios in the main text (see Figure 5). Subfigure titles above refer to differences from our "optimal" model in the lower right. **Upper left**: simulated data under a constant population size with fixed recombination rate. **Upper right**: simulated data under a 5-parameter exponential growth model with fixed recombination rate **Lower left**: Same as lower right but with a `max` permutation-invariant (PI) function. **Lower right**: Optimal model with `sum` permutation-invariant function, 5-parameter model, and HapMap recombination rates.

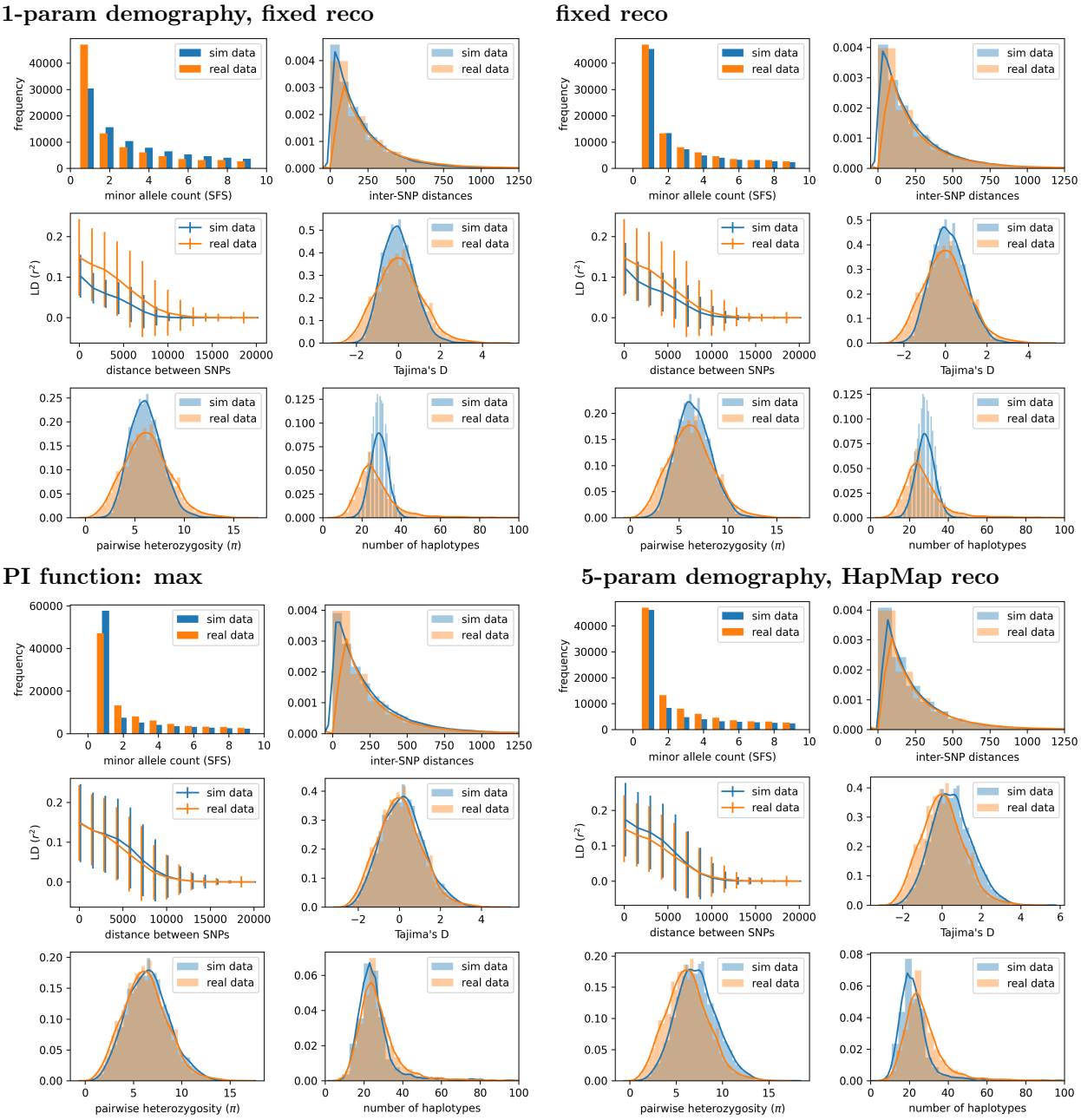

Figure S2: **CEU: single population model.** Summary statistic comparison between CEU data from the 1000 Genomes project and simulated data under the inferred parameters from the various scenarios in the main text (see Figure 5). Subfigure titles above refer to differences from our "optimal" model in the lower right. **Upper left**: simulated data under a constant population size with fixed recombination rate. **Upper right**: simulated data under a 5-parameter exponential growth model with fixed recombination rate **Lower left**: Same as lower right but with a `max` permutation-invariant (PI) function. **Lower right**: Optimal model with `sum` permutation-invariant function, 5-parameter model, and HapMap recombination rates.

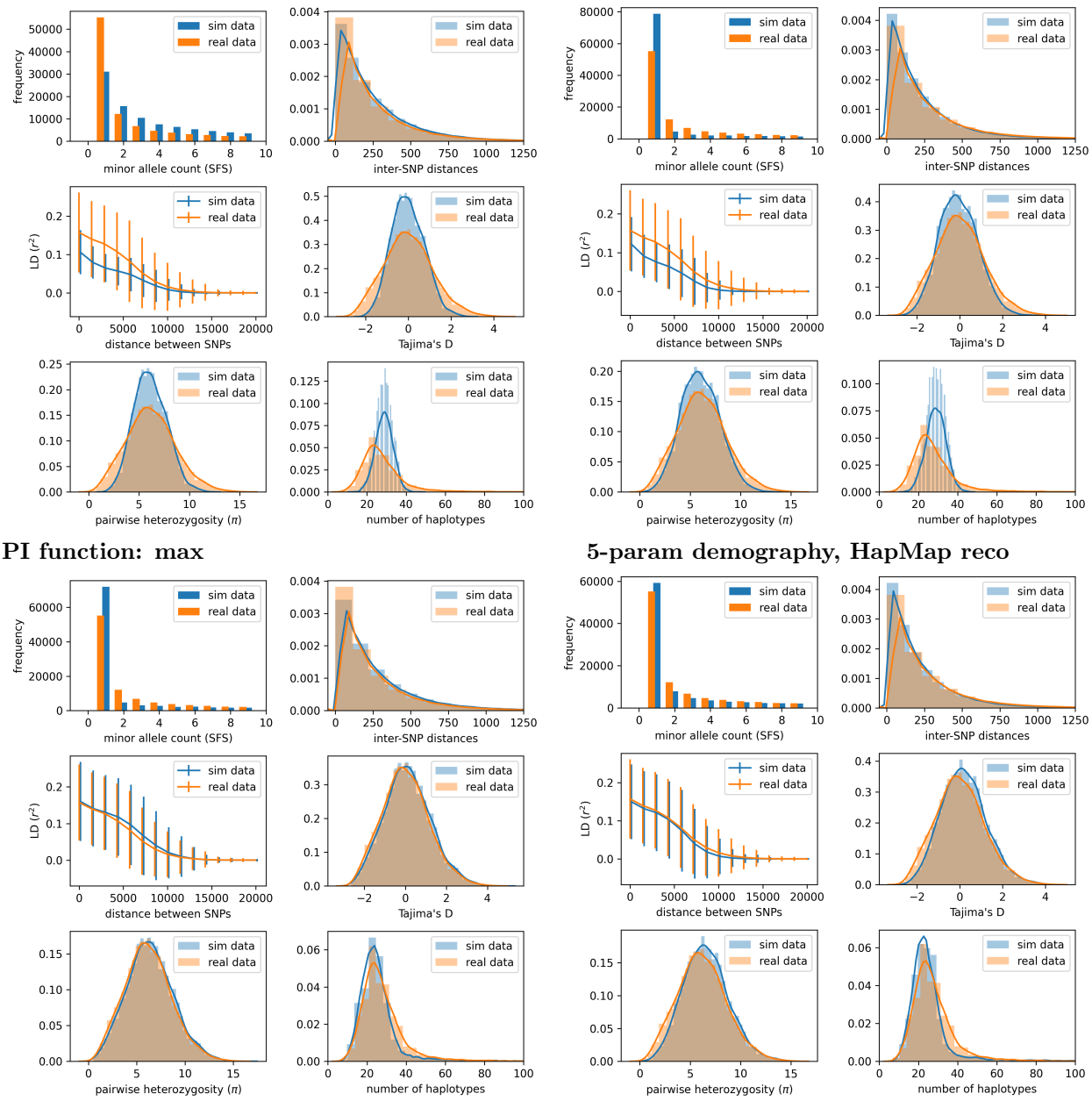

Figure S3: **CHB: single population model.** Summary statistic comparison between CHB data from the 1000 Genomes project and simulated data under the inferred parameters from the various scenarios in the main text (see Figure 5). Subfigure titles above refer to differences from our "optimal" model in the lower right. **Upper left**: simulated data under a constant population size with fixed recombination rate. **Upper right**: simulated data under a 5-parameter exponential growth model with fixed recombination rate **Lower left**: Same as lower right but with a `max` permutation-invariant (PI) function. **Lower right**: Optimal model with `sum` permutation-invariant function, 5-parameter model, and HapMap recombination rates.

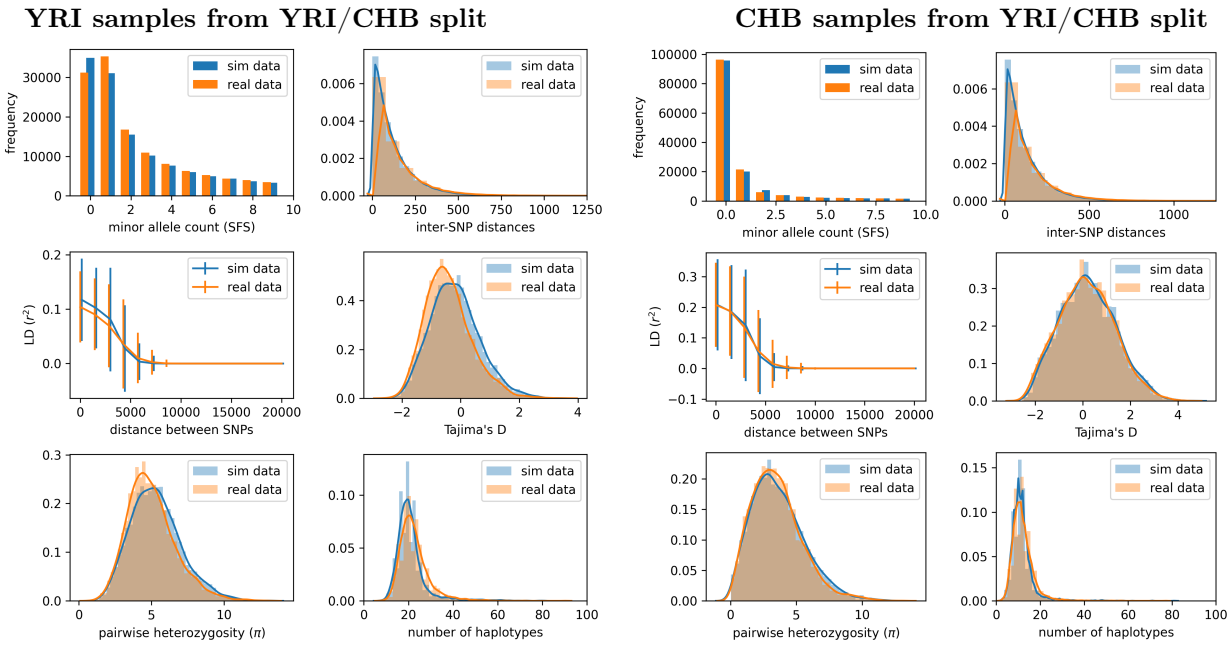

Figure S4: **YRI/CHB: two population model.** Summary statistic comparison real 1000 Genomes data and data simulated under the inferred parameters from Table 3 (second row). Left: statistics computed on YRI samples only. Right: statistics computed on CHB samples only. Note that we have non-segregating sites when considering each population separately, but not when we consider them together.

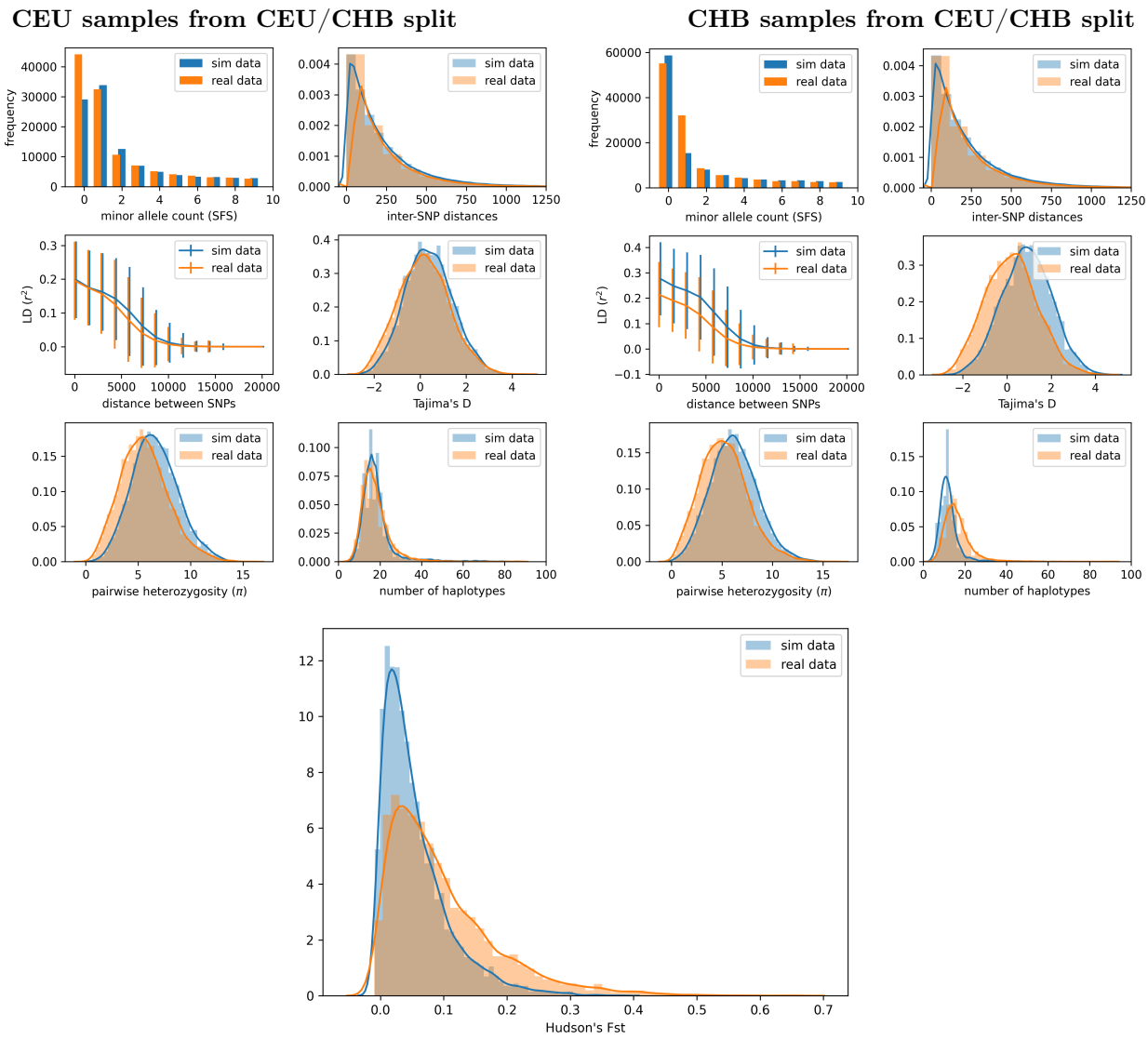

Figure S5: **CEU/CHB: two population model.** Summary statistic comparison real 1000 Genomes data and data simulated under the inferred parameters from the CEU/CHB split model Figure 3D. Left: statistics computed on CEU samples only. Right: statistics computed on CHB samples only. Lower panel: Fst, which we note is much less closely matched than for YRI/CEU or YRI/CHB. Note that we have non-segregating sites when considering each population separately, but not when we consider them together.