Analysis of simulated data: independent subpopulations and admixture population structures

Alejandro Ochoa and John D. Storey

2019-05-31

Introduction

In this R markdown file we reproduce the analysis of simulated datasets presented in our paper "F_{ST} and kinship for arbitrary population structures II: Method of moments estimators" by Ochoa and Storey. The analysis presented here relies entirely on simulated data and methods from publicly-available R packages.

For speed, and to increase sampling variance in these figures, we reduced the simulation to 1/10 of the individuals and 1/10 of the loci compared to the publication. This vignette takes about 7 seconds to compile on a single thread with a 3.6G processor and 31G of memory.

R package dependencies

To run this code, you will need the following packages:

```
library(bnpsd)         # to simulate genotypes from an admixed population
library(popkin)        # to estimate "population kinship" matrices
library(popkinsuppl)   # implements competing methods, available on OchoaLab GitHub
library(RColorBrewer) # for colors
library(MCMCpack)      # for rdirichlet, used in draw_subpop_sizes_dirichlet.R below
```

```
## Loading required package: coda
```

```
## Loading required package: MASS
```

```
## ##
## ## Markov Chain Monte Carlo Package (MCMCpack)
```

```
## ## Copyright (C) 2003-2019 Andrew D. Martin, Kevin M. Quinn, and Jong Hee Park
```

```
## ##
## ## Support provided by the U.S. National Science Foundation
```

```
## ## (Grants SES-0350646 and SES-0350613)
## ##
```

Let's load the following script, which defines the function `draw_subpop_sizes_dirichlet` that didn't make it to any of the packages.

```
source('scripts/draw_subpop_sizes_dirichlet.R')
```

Simulations

We consider two population structures:

- Independent subpopulations. This is the special case assumed by existing FST estimators. Shown for illustration and as a positive control.

- Admixed population. This is the more realistic scenario that only our approach `popkin` handles correctly.

Below we generate genotypes from each structure and study it in turn. Most of the functions involved in simulating the data are from the `bnpsd` package.

The dimensions of the data and the overall FST will be the same:

```
# number of individuals, 1/10 of paper
n_ind <- 100
# number of loci, 1/10 of paper
m_loci <- 30000
# number of subpopulations
k_subpops <- 10
# the FST of all individuals
Fst <- 0.1
```

Let's choose a single set of colors for the 10 subpopulations that appear in both of our simulations.

```
col_subpops <- brewer.pal(k_subpops, "Paired")
```

Simulation of independent subpopulations

Per-subpopulation FSTs

Let us choose the pattern of per-subpopulation FST values. In this case it is a ramp, so each subpopulation has an FST value proportional to its index.

```
inbr_subpops <- 1 : k_subpops
```

Next we ensure the mean `inbr_subpops` equals the desired FST, so it agrees with the WC and HudsonK estimators.

```
inbr_subpops <- inbr_subpops / mean(inbr_subpops) * Fst
```

Lastly, as a sanity check, we must make sure that the maximum FST does not exceed one. This is required to draw allele frequencies and genotypes correctly.

```
stopifnot( max(inbr_subpops) <= 1 )
```

This plots the resulting per-subpopulation FST values:

```
barplot(
    inbr_subpops,
    col = col_subpops,
    main = 'Intermediate subpop. diff.',
    ylab = expression(bold(F[ST])),
    xlab = 'Position in 1D geography'
)
```

Random subpopulation sample sizes

Next we randomly draw subpopulation sizes, constrained so that there are exactly `k_subpops` subpopulations and `n_ind` individuals in total. These sample sizes are draw from a rounded dirichlet (with small random adjustments so sum is correct), and ensuring that the smallest subpopulation is no smaller than a third of the mean size (the mean equaling `n_ind / k_subpops`).

```
labs <- draw_subpop_sizes_dirichlet(n_ind, k_subpops)
```

Our FST estimator does not use labels, but achieves the same effect through weights, which we compute using `weights_subpops` from the `popkin` package. This function computes weights such that every subpopulation has equal total weight, regardless of its sample size.

```
weights <- weights_subpops(labs)
```

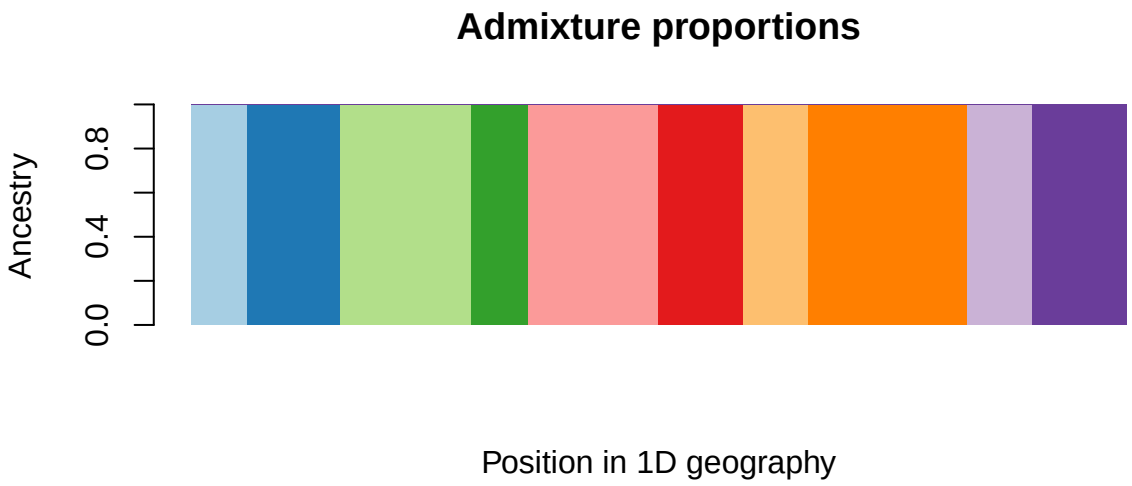
Subpopulation membership as trivial admixture proportions

We can then construct admixture proportions for this model. This is so we can use the generic admixture code to draw genotypes. However, these admixture proportions are trivial: for each individual, only one subpopulation has weight 1 (the one it belongs to as determined by `labs`), the rest have weight 0.

```
admix_proportions <- admix_prop_indep_subpops(labs)
```

This code plots the trivial admixture proportions:

```
barplot(
    t(admix_proportions),
    col = col_subpops,
    xlab = 'Position in 1D geography',
    ylab = "Ancestry",
    border = NA,
    space = 0,
    main = 'Admixture proportions'
)
```



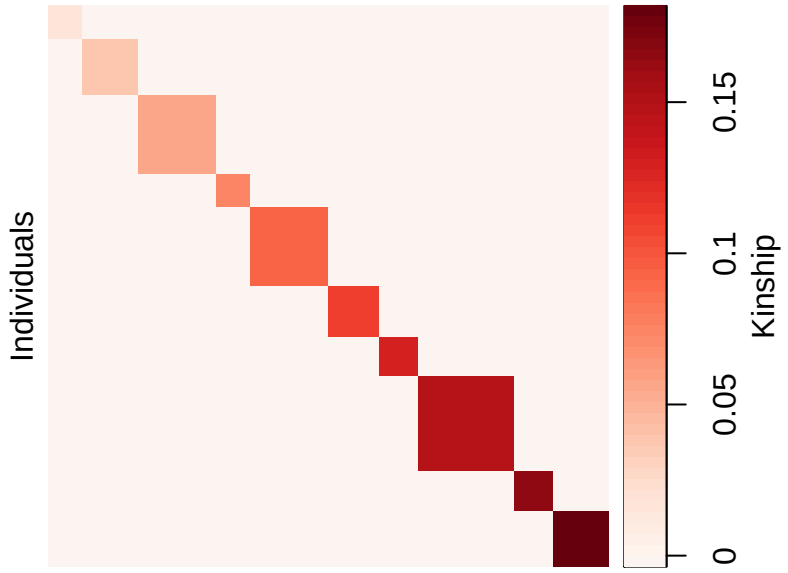
Coancestry (kinship) matrix

We use the previous population structure parameters to construct the coancestry matrix of the simulation.

```
coancestry <- coanc_admix(admix_proportions, inbr_subpops)
```

We use the `plot_popkin` function from `popkin` to visualize the coancestry matrix. The coancestry matrix is just like a kinship matrix, but already contains inbreeding coefficients along the diagonal. Therefore, although `inbr_diag` (from `popkin`) is usually needed for kinship matrices, it is not needed here.

```
# set outer margins
par(oma = c(0, 1.5, 0, 3))
# don't need inner margins if there are no labels
par(mar = c(0, 0, 0, 0) + 0.2)
plot_popkin( coancestry )
```



Draw genotypes

Now we draw genotypes from this model.

```
X <- draw_all_admix(admix_proportions, inbr_subpops, m_loci)$X
```

Kinship estimate comparisons

Here we will generate several kinship estimates and compare them to the true kinship matrix and to a limit we calculated.

The function `kinship_std_limit` from `popkinsuppl` calculates the limit of the standard kinship estimator. This is for the ratio-of-means version (ROM), the only version that has a limit that can be calculated. Note that to be correct, the coancestry matrix has to be converted first into a proper kinship matrix:

```
# this scales the diagonal values correctly
kinship <- coanc_to_kinship(coancestry)
# calculate limit of kinship estimates
kinship_biased_limit <- kinship_std_limit( kinship )
```

Now we obtain the popkin estimate. We use the genotypes and the subpopulation labels (only used to estimate the minimum kinship value to set to zero):

```
kinship_popkin <- popkin( X, labs )
```

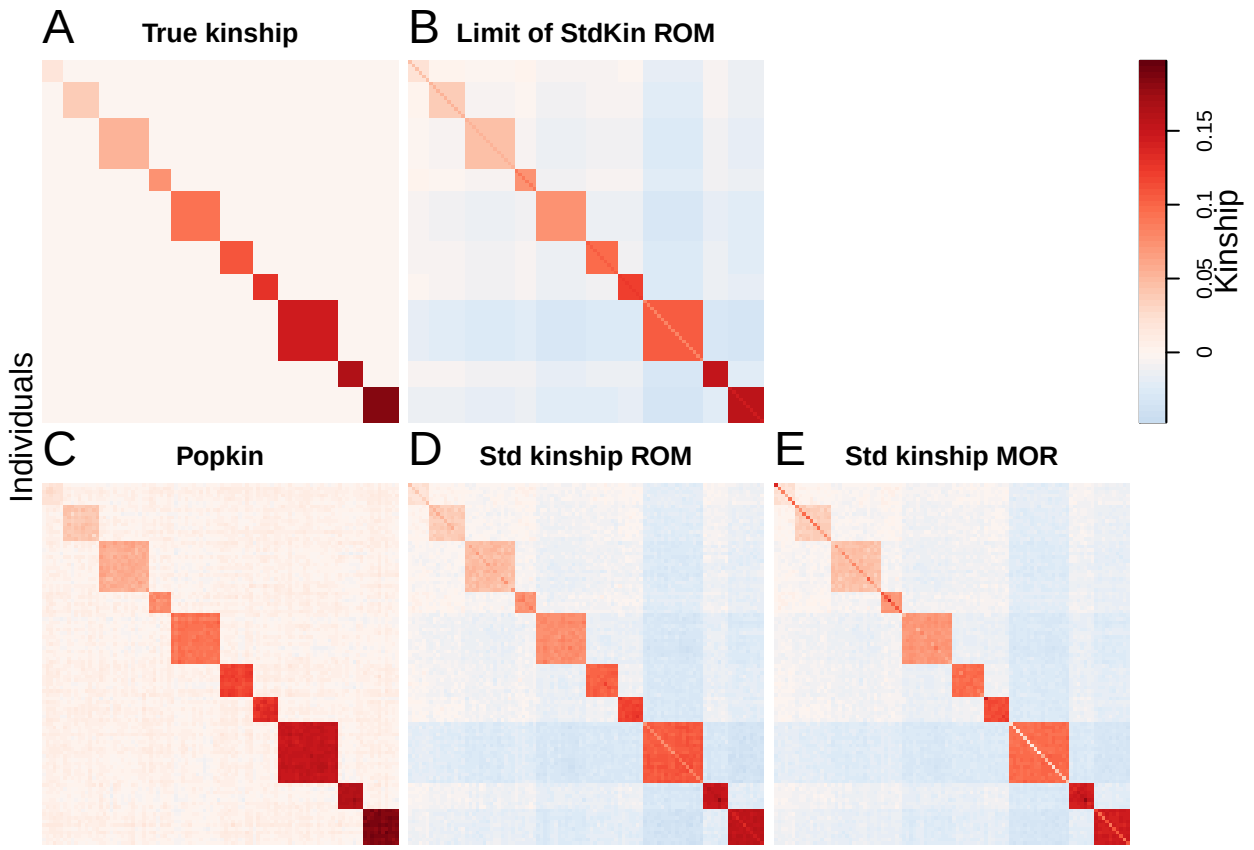
We calculate two standard kinship estimates: ROM and MOR (mean-of-ratios) versions. The ROM version is the focus of our simulation studies, as it has a known limit. However, MOR is more commonly used in the literature, and that version is used in the real human analyses.

```
kinship_std_rom <- kinship_std( X )
kinship_std_mor <- kinship_std( X, mean_of_ratios = TRUE )
```

Gather all estimates into a list and plot

```
# list of kinship matrices.
# NULL leaves panel 3 blank
kinship_list <- list(
    kinship,
    kinship_biased_limit,
    NULL,
    kinship_popkin,
    kinship_std_rom,
    kinship_std_mor
)
# names for plot.
# Note that NULL panel is skipped (works fine)
kinship_names <- c(
    'True kinship',
    'Limit of StdKin ROM',
    'Popkin',
    'Std kinship ROM',
    'Std kinship MOR'
)
# set reasonable margins
par(oma = c(0, 1.5, 0, 3))
par(mar = c(0, 0, 2, 0) + 0.2)
# plot!
plot_popkin(
    kinship = inbr_diag( kinship_list ),
    titles = kinship_names,
```

```
    layout_rows = 2
)
```



FST estimates

The true FST is a parameter set in the simulation,

```
Fst
```

```
## [1] 0.1
```

which, for independent subpopulations, equals the mean of the per-subpopulation FST values:

```
mean( inbr_subpops )
```

```
## [1] 0.1
```

FST is also given by our generalized definition, as the weighted mean inbreeding of all individuals, where the weights balance subpopulation sizes:

```
# direct calculation of weighted mean inbreeding
drop( inbr( kinship ) %*% weights )
```

```
## [1] 0.1
```

```
# popkin implementation (same but with additional validation steps)
fst(kinship, weights)
```

```
## [1] 0.1
```

Note that in this simulations all individuals are "locally outbred" (there are no individuals whose parents are siblings, first cousens, etc), which is an important requirement for the above value to actually be FST (otherwise it is analogous to Wright's FIT, the total inbreeding coefficient).

Above, all those equivalent calculations of FST were in terms of the true parameters of the simulation. Now we proceed to estimate FST from the genotype data, without knowledge of the true kinship matrix.

In popkin, FST is estimated just as in the last step, which corresponds to the generalized FST definition, but with the true kinship matrix replaced by the popkin estimate (which was obtained earlier from genotypes and the subpopulation labels only):

```
# calculate and store
Fst_popkin <- fst(kinship_popkin, weights)
# inspect
Fst_popkin
```

```
## [1] 0.102914
```

For comparison, we shall compute the FST estimate obtained from the standard kinship matrix estimate, which is not optimized for this use but illustrates its overall bias that the popkin estimate overcomes:

```
Fst_std <- fst(kinship_std_rom, weights)
Fst_std
```

```
## [1] 0.09099766
```

The most appropriate FST estimator for this exact simulation is the "HudsonK" FST estimator (a generalized form for K subpopulations (Ochoa and Storey 2016) of the original "Hudson" FST estimator for two subpopulations of Bhatia et al (2013)). This estimator requires independent subpopulations (zero kinship between subpopulations), and handles different per-subpopulation FST values (`inbr_subpops` here) and different sample sizes per subpopulation. We implemented this estimator in the `popkinsuppl` package:

```
Fst_hudson_k <- fst_hudson_k( X, labs = labs )$fst
Fst_hudson_k
```

```
## [1] 0.1003068
```

The Weir-Cockerham (WC) 1984 estimator achieves a similar result but technically requires every per-subpopulation FST to be the same, so in this setting it is expected to have a small bias:

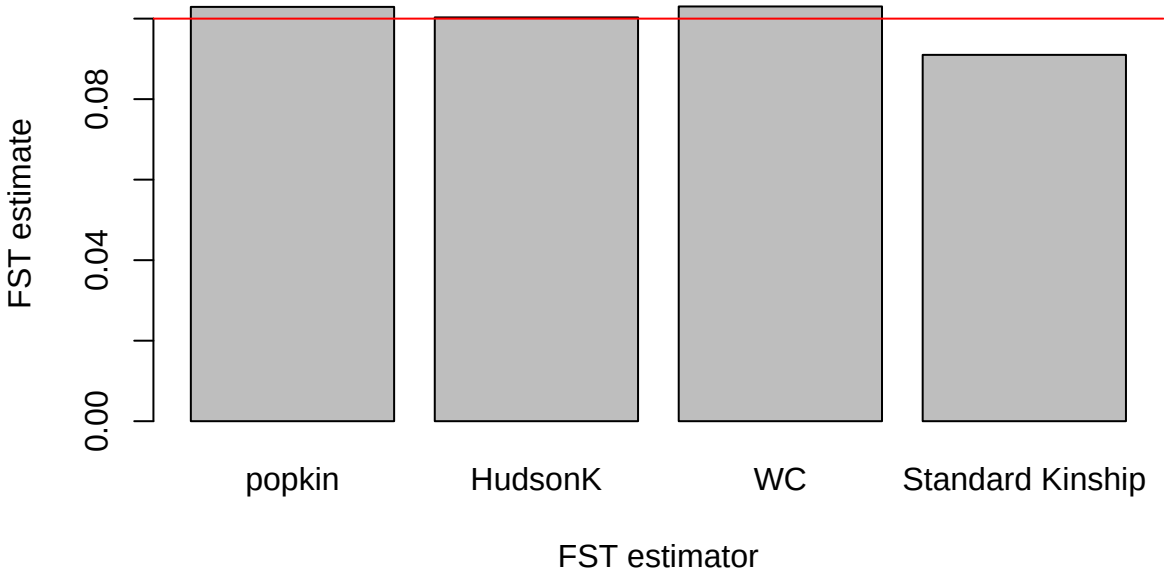
```
Fst_wc <- fst_wc( X, labs = labs )$fst
Fst_wc
```

```
## [1] 0.1030218
```

Here is a simple plot that compares these estimates to the true FST:

```
# vector of estimates
Fst_estimates <- c(
    Fst_popkin,
    Fst_hudson_k,
    Fst_wc,
    Fst_std
)
# vector of names
Fst_names <- c(
    'popkin',
    'HudsonK',
    'WC',
    'Standard Kinship'
)
```

```r
# plot estimates
barplot(
    Fst_estimates,
    names.arg = Fst_names,
    xlab = 'FST estimator',
    ylab = 'FST estimate',
    ylim = c(0, max(Fst, Fst_popkin, Fst_hudson_k, Fst_wc))
)
# overlay a red line that shows where the true value lies
abline(h = Fst, col = 'red')
```



This concludes our positive control, where we demonstrate that our implementations of the existing FST estimators work as advertised on their assumed model of independent subpopulations. Although popkin makes considerably weaker assumptions, it performs very well in this setting too. We shall see in the next section that only the popkin FST estimator is accurate when there are no independent subpopulations, such as for differentially-admixed individuals.

Simulation of an admixed population

Per-subpopulation FSTs, admixture proportions

The more general admixture model also starts from independent subpopulations that serve as the ancestral subpopulations for the admixed individuals. We again define the per-subpopulations FST values as a ramp (each subpopulation has an FST proportional to its index. For now this will be in the wrong scale, but our code fixes it later as it fits other parameters of the simulation.

```r
inbr_subpops <- 1 : k_subpops
```

Here we declare the desired "bias coefficient", which is the ratio of the final FST to the mean coancestry coefficient. This parameter approximately equals the bias of that existing kinship and FST estimator will have on this dataset.

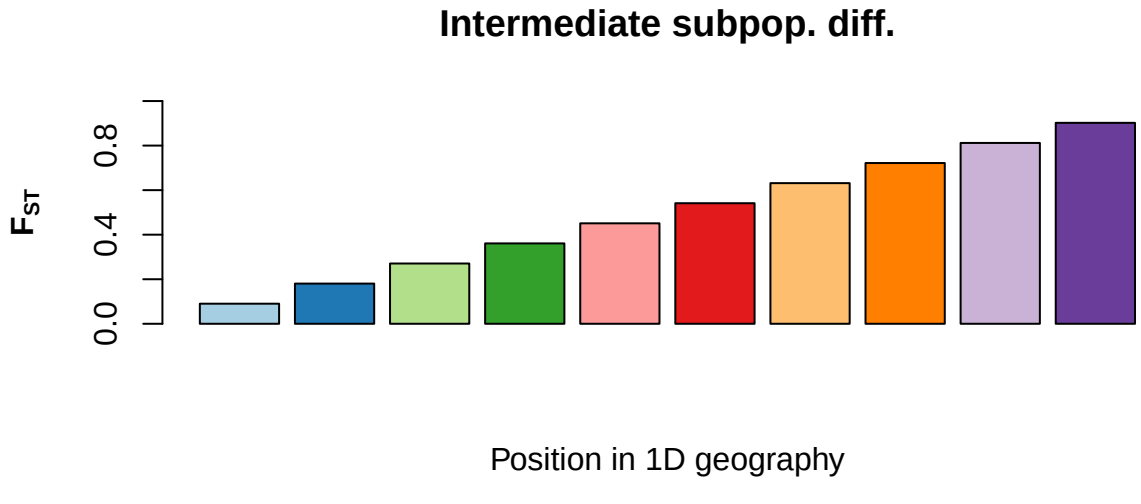
```r
bias_coeff <- 0.5
```

This function draws the admixture proportions that have the desired bias coefficient, generalized FST, and rescales the per-subpopulation FST values (`inbr_subpops` below).

```
obj <- admix_prop_1d_linear(
    n_ind = n_ind,
    k_subpops = k_subpops,
    bias_coeff = bias_coeff,
    coanc_subpops = inbr_subpops,
    fst = Fst
)
# admixture proportions
admix_proportions <- obj$admix_proportions
# rescaled FST vector for intermediate subpopulations
inbr_subpops <- obj$coanc_subpops
# spread off intermediate subpopulations
sigma <- obj$sigma
```

This plots the resulting per-subpopulation FST values:

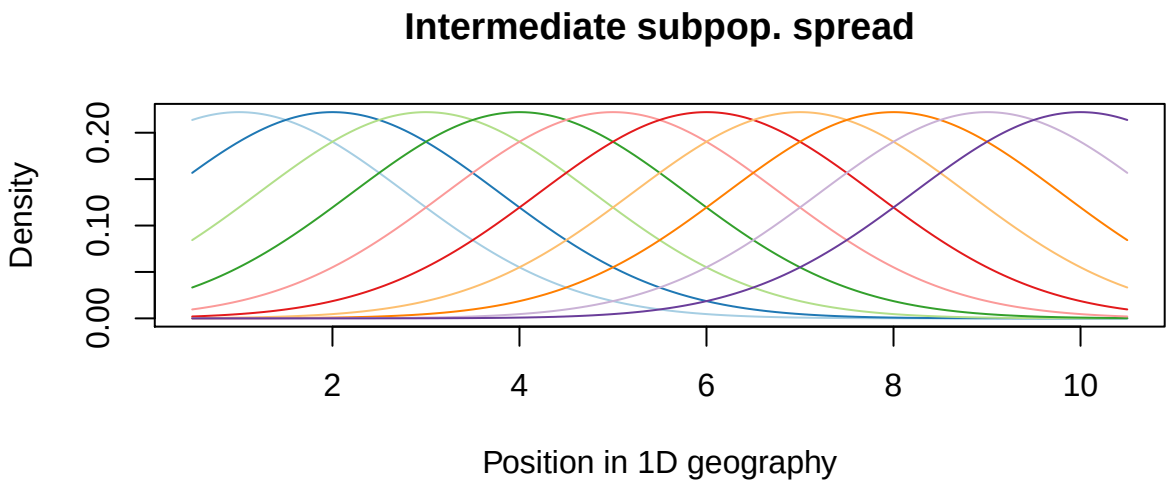
```
barplot(
    inbr_subpops,
    col = col_subpops,
    main = 'Intermediate subpop. diff.',
    ylab = expression(bold(F[ST])),
    xlab = 'Position in 1D geography',
    ylim = c(0,1)
)
```



This constructs and plots the densities of the intermediate subpopulations implied in our 1D geography admixture scenario:

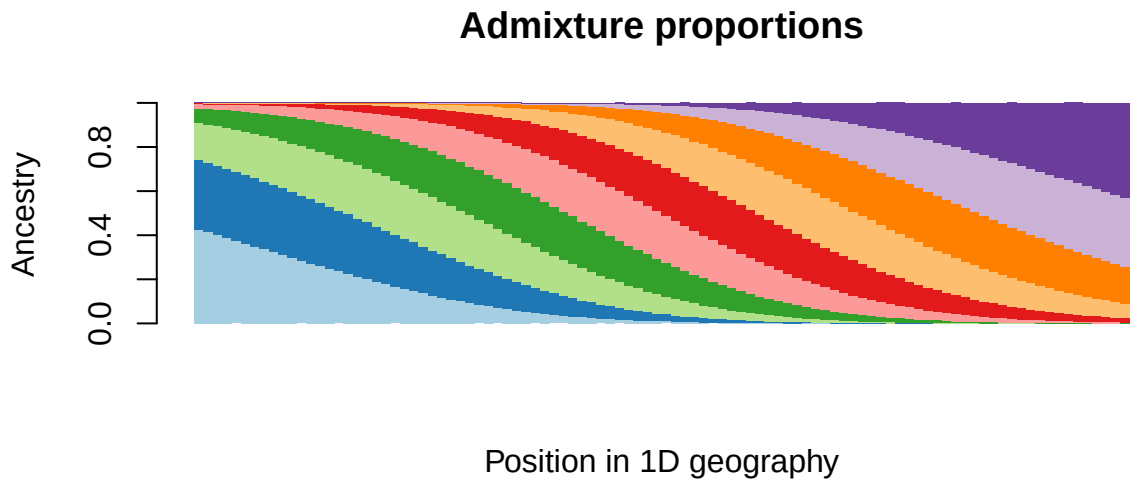
```
# set some resolution on the x axis of 1000 points
# and a range of 0.5 to k_subpops + 0.5
xs <- 0.5 + (0 : 999) / 999 * k_subpops
# this creates figure for first subpopulation,
# which sets plotting ranges right
ys <- dnorm(xs, mean = 1, sd = sigma)
plot(
    xs,
    ys,
    col = col_subpops[1],
    type = 'l',
    xlab = 'Position in 1D geography',
```

```
    ylab = 'Density',
    main = 'Intermediate subpop. spread'
)
# add curves for the remainder of the subpopulations
for (u in 2 : k_subpops) {
  ys <- dnorm(xs, mean = u, sd = sigma)
  lines(xs, ys, col = col_subpops[u])
}
```



This code plots the resulting admixture proportions:

```
barplot(
    t(admix_proportions),
    col = col_subpops,
    xlab = 'Position in 1D geography',
    ylab = "Ancestry",
    border = NA,
    space = 0,
    main = 'Admixture proportions'
)
```



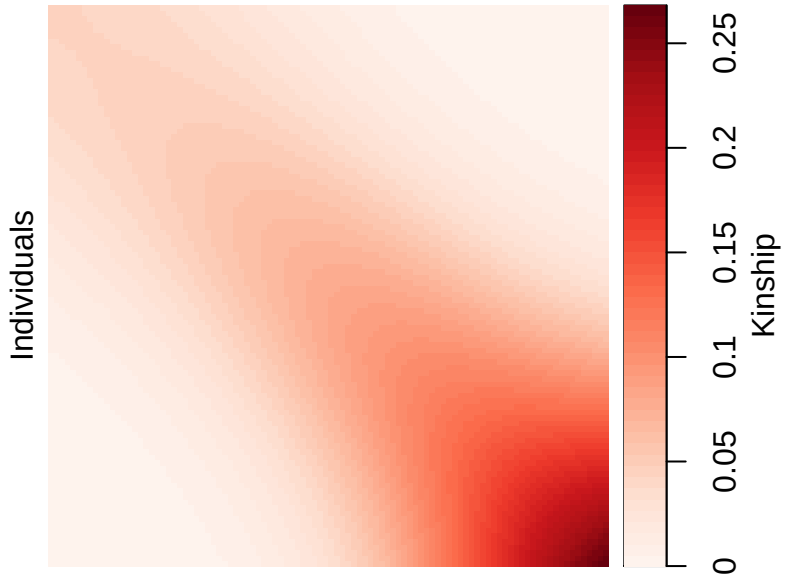
Coancestry (kinship) matrix

We use the previous population structure parameters to construct the coancestry matrix of the simulation.

```
coancestry <- coanc_admix(admix_proportions, inbr_subpops)
```

We use the `plot_popkin` function from `popkin` to visualize the coancestry matrix (see earlier notes for more info).

```
par(oma = c(0, 1.5, 0, 3))
par(mar = c(0, 0, 0, 0) + 0.2)
plot_popkin( coancestry )
```



Draw genotypes

Lastly, we draw genotypes from this model.

```
out <- draw_all_admix(admix_proportions, inbr_subpops, m_loci)
# extract parts
# genotypes
X <- out$X
# true ancestral allele frequencies, used in some limited demonstrations
p_anc <- out$p_anc
```

Subpopulation assignments

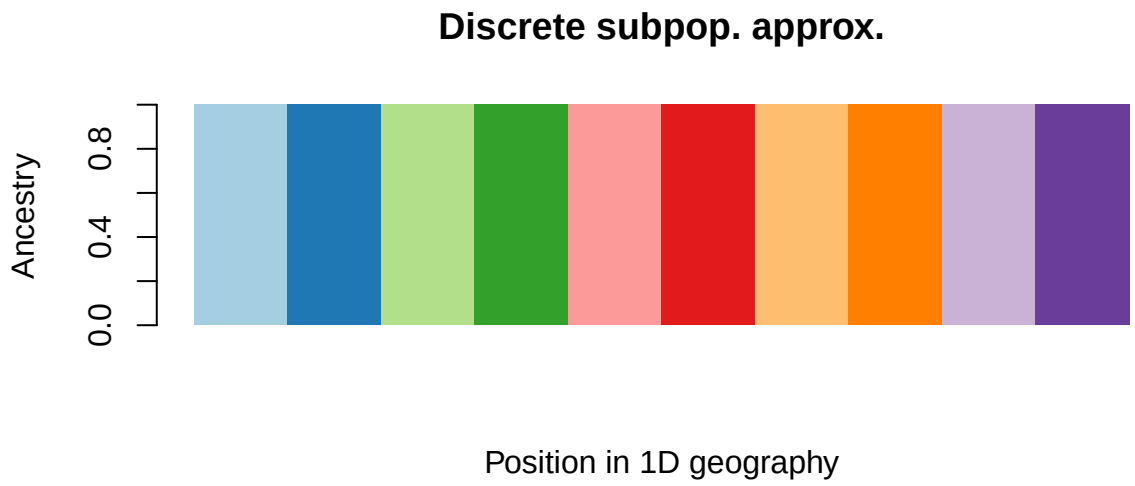
Here we construct subpopulation labels for our admixture model, based on the majority admixture proportion. This is to approximate the independent subpopulations model, in order to apply existing FST estimators (WC and HudsonK). This also aids `popkin` in estimating the minimum kinship value.

```
# define abstract function
make_labs_max_admix_prop <- function(admix_proportions) {
    # find max per row, make sure only first element is used in case of ties
    # returns desired vector of indexes
    apply(admix_proportions, 1, function(x) which( x == max(x) )[1] )
}
```

```
# apply function to our particular simulation
labs <- make_labs_max_admix_prop(admix_proportions)
```

Here is the visualization of these artificial subpopulation assignments:

```
barplot(
    rep.int(1, n_ind),
    col = col_subpops[labs],
    xlab = 'Position in 1D geography',
    ylab = "Ancestry",
    border = NA,
    space = 0,
    main = 'Discrete subpop. approx.'
)
```



Kinship estimate comparisons

Here we repeat our kinship estimate analysis (first performed for independent subpopulations above), comparing our new approach to the standard kinship estimator and its calculated limit. The text is reproduced for clarity.

The function `kinship_std_limit` from `popkinsuppl` calculates the limit of the standard kinship estimator. This is for the ratio-of-means version (ROM), the only version that has a limit that can be calculated. Note that to be correct, the coancestry matrix has to be converted first into a proper kinship matrix:

```
# this scales the diagonal values correctly
kinship <- coanc_to_kinship(coancestry)
# calculate limit of kinship estimates
kinship_biased_limit <- kinship_std_limit( kinship )
```

Now we obtain the popkin estimate. We use the genotypes and the subpopulation labels (only used to estimate the minimum kinship value to set to zero):

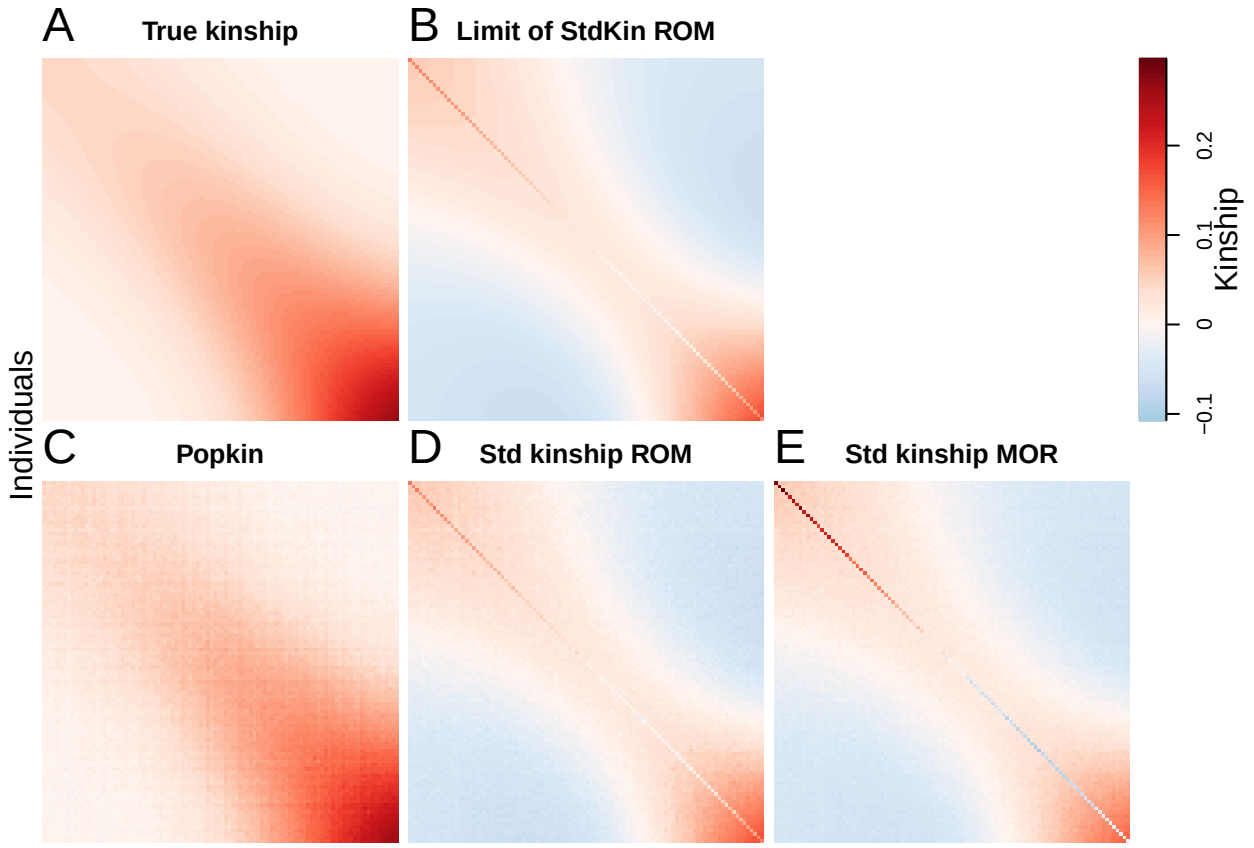
```
kinship_popkin <- popkin( X, labs )
```

We calculate two standard kinship estimates: ROM and MOR (mean-of-ratios) versions. The ROM version is the focus of our simulation studies, as it has a known limit. However, MOR is more commonly used in the literature, and that version is used in the real human analyses.

```
kinship_std_rom <- kinship_std( X )
kinship_std_mor <- kinship_std( X, mean_of_ratios = TRUE )
```

Gather all estimates into a list and plot

```
# list of kinship matrices.
# NULL leaves panel 3 blank
kinship_list <- list(
    kinship,
    kinship_biased_limit,
    NULL,
    kinship_popkin,
    kinship_std_rom,
    kinship_std_mor
)
# names for plot.
# Note that NULL panel is skipped (works fine)
kinship_names <- c(
    'True kinship',
    'Limit of StdKin ROM',
    'Popkin',
    'Std kinship ROM',
    'Std kinship MOR'
)
# set reasonable margins
par(oma = c(0, 1.5, 0, 3))
par(mar = c(0, 0, 2, 0) + 0.2)
# plot!
plot_popkin(
    kinship = inbr_diag( kinship_list ),
    titles = kinship_names,
    layout_rows = 2
)
```



FST estimates

Here we reproduce the earlier FST comparisons performed on the independent subpopulations, with the important caveat that the assumptions of existing FST estimators are not met at all in this admixture setting.

The true FST is a parameter set in the simulation,

```
Fst
```

```
## [1] 0.1
```

This value corresponds to the generalized FST of the admixed individuals. In this simulation we used uniform weights, so the calculations simplify:

```
# direct calculation of mean inbreeding
mean( inbr( kinship ) )
```

```
## [1] 0.1
```

```
# popkin implementation (same but with additional validation steps)
fst(kinship)
```

```
## [1] 0.1
```

Note that in this simulations all individuals are "locally outbred" (there are no individuals whose parents are siblings, first cousens, etc), which is an important requirement for the above value to actually be FST (otherwise it is analogous to Wright's FIT, the total inbreeding coefficient).

Unlike the earlier independent subpopulations simulation, here the generalized FST does not equal the mean of the per-subpopulation FST values for the (unadmixed) intermediate subpopulations, which must be larger

since admixture decreases differentiation:

```
mean( inbr_subpops )
```

```
## [1] 0.4961805
```

The above calculations of FST were in terms of the true parameters of the simulation. Now we proceed to estimate FST from the genotype data, without knowledge of the true kinship matrix.

In popkin, FST is estimated just as in the last step, which corresponds to the generalized FST definition, but with the true kinship matrix replaced by the popkin estimate (which was obtained earlier from genotypes and the subpopulation labels only):

```
# calculate and store
Fst_popkin <- fst(kinship_popkin)
# inspect
Fst_popkin
```

```
## [1] 0.09862863
```

For comparison, we shall compute the FST estimate obtained from the standard kinship matrix estimate, which is not optimized for this use but illustrates its overall bias that the popkin estimate overcomes:

```
Fst_std <- fst(kinship_std_rom)
Fst_std
```

```
## [1] 0.04668108
```

Now we apply the most common existing FST estimators. First is the "HudsonK" FST estimator (a generalized form for K subpopulations (Ochoa and Storey 2016) of the original "Hudson" FST estimator for two subpopulations of Bhatia et al (2013)). This estimator requires independent subpopulations, which do not exist for our admixed individuals, so its model is misspecified:

```
Fst_hudson_k <- fst_hudson_k( X, labs = labs )$fst
Fst_hudson_k
```

```
## [1] 0.05631891
```

The Weir-Cockerham (WC) 1984 estimator is similarly misspecified in this setting:

```
Fst_wc <- fst_wc( X, labs = labs )$fst
Fst_wc
```

```
## [1] 0.05630618
```

Our calculations show that the standard kinship estimate, and to a lesser extent both the WC and HudsonK estimators, have a bias that approaches the following value (as the number of loci goes to infinity, and for WC and HudsonK also as the number of subpopulations goes to infinity), given by the true mean kinship of the data (in this case with uniform weights):

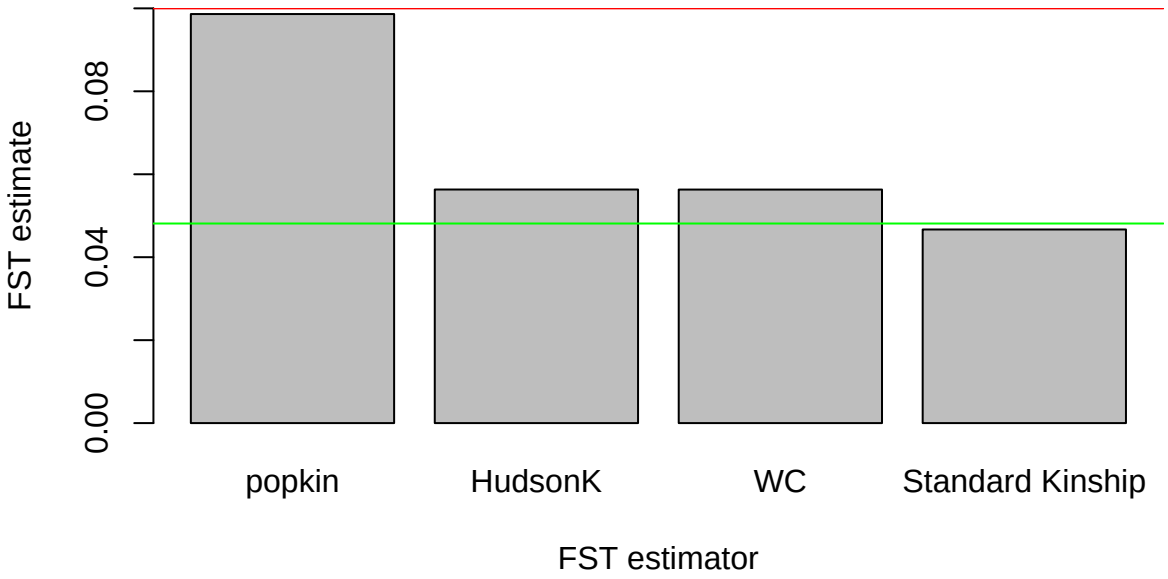
```
Fst_limit_indep_subpops <- ( Fst - mean(kinship) ) / ( 1 - mean(kinship) )
Fst_limit_indep_subpops
```

```
## [1] 0.04812269
```

Here is a simple plot that compares these estimates to the true FST and the limit we calculated for the existing estimators:

```
# vector of estimates
Fst_estimates <- c(
    Fst_popkin,
    Fst_hudson_k,
    Fst_wc,
```

```
    Fst_std
)
# vector of names
Fst_names <- c(
    'popkin',
    'HudsonK',
    'WC',
    'Standard Kinship'
)
# plot estimates
barplot(
    Fst_estimates,
    names.arg = Fst_names,
    xlab = 'FST estimator',
    ylab = 'FST estimate',
    ylim = c(0, max(Fst, Fst_popkin, Fst_hudson_k, Fst_wc))
)
# overlay a red line that shows where the true value lies
abline(h = Fst, col = 'red')
# and a green line that shows the limit of the biased estimators
abline(h = Fst_limit_indep_subpops, col = 'green')
```



This demonstrates that only the popkin estimate remains accurate in data without independent subpopulations, and that the existing estimators all have essentially the same bias. We found that their bias is due to treating most kinship values as being equal to zero, which is not true in this simulation and is very unlikely in real datasets (see the Human Origins and 1000 Genomes Hispanics analysis presented in the other vignettes of this repository).