Analysis of Hispanics in 1000 Genomes data

Alejandro Ochoa and John D. Storey

2019-05-29

Introduction

In this R markdown file we demonstrate the basic analysis of Hispanic individuals from the 1000 Genomes Project (TGP) presented in our paper "New kinship and FST estimates reveal higher levels of differentiation in the global human population" by Ochoa and Storey. We rely exclusively on the data files provided in this repository (https://github.com/StoreyLab/human-differentiation-manuscript).

This vignette takes about 18 minutes to compile on a single thread with a 3.6G processor and 31G of memory. The slowest steps are estimation of two kinship matrices (our new `popkin` method, and the existing Standard Kinship formula), and the two existing FST methods (Weir-Cockerham and Hudson K).

R package dependencies

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

```
library(BEDMatrix)    # to load genotype matrices
library(popkin)       # to estimate "population kinship" matrices
library(popkinsuppl)  # implements competing methods, available on OchoaLab GitHub
library(genio)        # to parse FAM file
library(readr)        # to parse admixture proportions table
library(seriation)    # to automatically order individuals given kinship estimate
library(RColorBrewer) # for colors
```

Estimate the kinship matrix

Load the genotype matrix (smartly, actually doesn't load into memory):

```
X <- BEDMatrix('data/hispanics', simple_names = TRUE)
```

```
## Extracting number of samples and rownames from hispanics.fam...
```

```
## Extracting number of variants and colnames from hispanics.bim...
```

This function from `genio` loads individual annotations (in plink FAM format), including subpopulations.

```
fam <- read_fam('data/hispanics')
```

```
## Reading: data/hispanics.fam
```

Now estimate the kinship matrix. It doesn't make sense to use the TGP subpopulations here, so we'll ignore them as we estimate the kinship matrix. This step takes about 7 minutes.

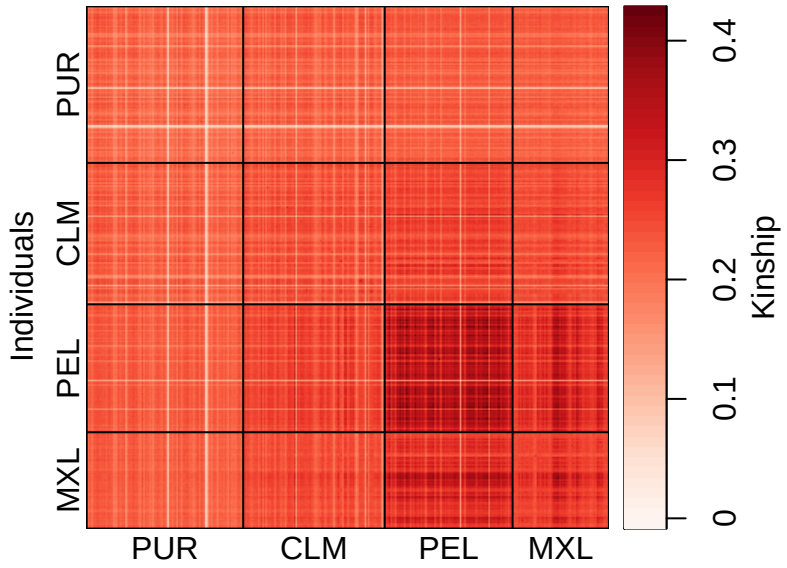
```
kinship <- popkin(X)
```

As a first pass, let's order individuals grouping by subpopulation. This code achieves that, using the following custom order:

```
# desired order
# Puerto Ricans, Colombians, Peruvians, Mexican-Americans
subpop_order <- c('PUR', 'CLM', 'PEL', 'MXL')
# indexes to reorder individuals
indexes <- order(match(fam$fam, subpop_order))
# keep a copy of fam in the original order (used by FST estimators later)
fam_orig <- fam
# reorder kinship matrix (both dimensions)
kinship <- kinship[indexes, indexes]
# reorder individual annotations (rows only)
fam <- fam[indexes, ]
```

We are ready to visualize the estimated kinship matrix:

```
# set outer margin for axis labels (left and right are non-zero)
par(oma = c(0, 1.5, 0, 3))
# set inner margin for subpopulation labels (bottom and left are non-zero), add padding
par(mar = c(1, 1, 0, 0) + 0.2)
# now plot!
plot_popkin(
    kinship = inbr_diag(kinship),
    labs = fam$fam
)
```



Order individuals using seriation

Here we call the `seriate` function from the package `seriation` to order individuals. This function arranges individuals by grouping the most similar closer to each other, and the most dissimilar far from each other. In terms of the kinship matrix, this approach will concentrate the largest values on the diagonal and the lowest values farthest from the diagonal.

Here is the function wrapper we will use:

```
# returns order for a kinship matrix
order_seriate <- function(kinship) {
    # turn similarity (kinship) into dissimilarity (distances)
```

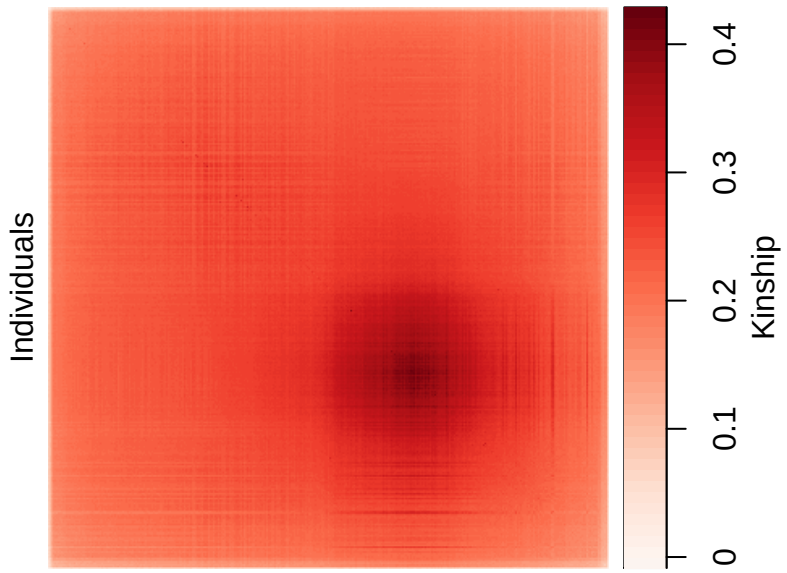
```
    # start by negating
    distance <- - kinship
    # shift so minimum value is zero
    distance <- distance - min(distance)
    # make distance object, expected as input
    distance <- as.dist(distance)

    # perform desired optimization
    # "ARSA" (anti-Robinson) method
    seriation_object <- seriate(distance, method = 'ARSA')
    # the order we want with this other seriation function
    indexes <- get_order( seriation_object )

    # reverse order depending on slope
    # (for consistency)
    # (every `seriate` run is random and order is reversed often)
    y <- diag(kinship)[indexes]
    x <- 1 : length(y)
    m <- coef(lm(y ~ x))[2] # this is the slope of interest (from linear fit)
    if (m < 0) { # flip negative slope cases only
        indexes <- rev( indexes )
    }

    # done, return order
    return( indexes )
}
```

Let's apply it to our kinship matrix, reordering everything we have

```
# get order
indexes <- order_seriate(kinship)
# reorder kinship matrix (both dimensions)
kinship <- kinship[indexes, indexes]
# reorder individual annotations (rows only)
fam <- fam[indexes, ]
```

Let's plot the new order! Since the subpopulation labels are all scrambled, let's not use them now (we will plot them using another method shortly).

```
par(oma = c(0, 1.5, 0, 3))
# don't need margins if there are no labels
par(mar = c(0, 0, 0, 0) + 0.2)
# now plot!
plot_popkin(
    kinship = inbr_diag(kinship)
)
```



Rescaling the kinship estimates

Seriation has placed the most unrelated individuals at the extremes of the current order. This allows us to get a better estimate of the baseline kinship value to be treated as zero. The code below chooses two individuals from one extreme, and two other individuals at the other extreme, and via `rescale_popkin` sets their mean kinship value as the new zero value.

```
# first construct dummy populations
# total number of individuals
n_ind <- nrow(fam)
# number of individual in extreme subpopulations
n_subpop_dummy <- 2
# most individuals are assigned pop 1
fam$subpop_dummy <- 1
# the first extreme population is pop 2
fam$subpop_dummy[ 1 : n_subpop_dummy ] <- 2
# the other extreme population is pop 3
fam$subpop_dummy[ n_ind + 1 - ( 1 : n_subpop_dummy ) ] <- 3

# report IDs of individuals in these extreme subpopulations
fam$id[ fam$subpop_dummy == 2 ]
```

```
## [1] "HG01108" "HG01242"
```

```
fam$id[ fam$subpop_dummy == 3 ]
```

```
## [1] "HG01551" "HG01241"
```

```
# before rescaling, note that the minimum kinship value was exactly zero
range( inbr_diag(kinship) )
```

```
## [1] -0.007938456  0.428759027
```

```
# also get FST (to see change later)
fst( kinship )
```

```
## [1] 0.2574028
```

```
# rescale to have a better kinship estimate
kinship <- rescale_popkin(kinship, subpops = fam$subpop_dummy)

# after rescaling, the minimum kinship value is slightly negative
# (this is random noise, estimates overall will be less biased now)
# all values are smaller now
range( inbr_diag(kinship) )
```

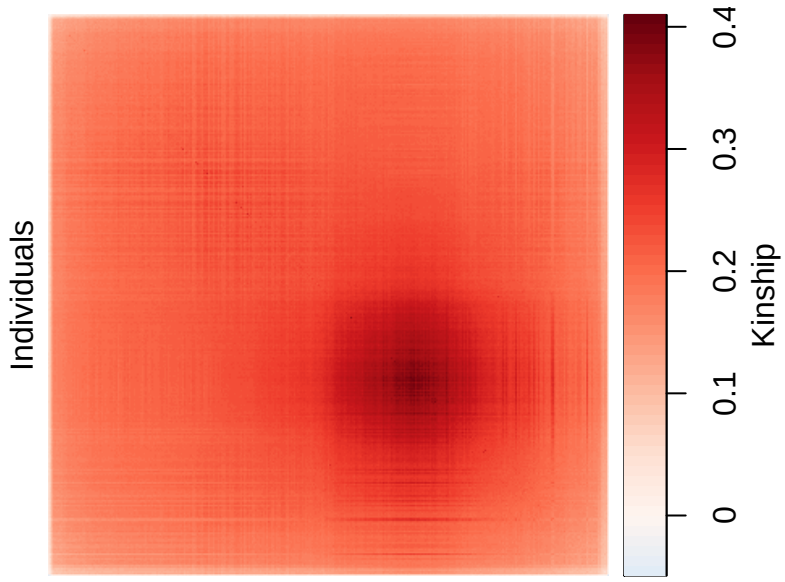
```
## [1] -0.04154273  0.40971408
```

```
# FST got noticeably smaller too
fst( kinship )
```

```
## [1] 0.2326449
```

This plot shows that the kinship matrix did not change except (slightly) in scale:

```
par(oma = c(0, 1.5, 0, 3))
par(mar = c(0, 0, 0, 0) + 0.2)
# now plot!
plot_popkin(
    kinship = inbr_diag(kinship)
)
```



Colors for geographic and admixture subpopulations

Here we choose colors for two classes of subpopulations: the sampling location labels (PUR, CLM, PEL, MXL) and admixture subpopulations (AFR, EUR, AMR, which will be used later in this document). We obtain colors jointly for both classes so that they don't clash and lead to confusion:

```
# number of admixture subpopulations
k_admix <- 3
# number of geographic subpopulations
n_subpops <- length(subpop_order)
# get a joint set of colors that don't clash
colors_all <- brewer.pal( n_subpops + k_admix, "Set3" )
```

```
# now we split the colors for both classes
# first set of colors are for the geographic labels
colors_subpops <- colors_all[ 1 : n_subpops ]
# the rest are for the admixture subpopulations
colors_admix <- setdiff(colors_all, colors_subpops)
```

Marking subpopulations with colors on the sides

This code assigns to every individual the color of its subpopulation. We store the colors in the FAM tibble of individual annotations.

```
fam$col <- colors_subpops[match(fam$fam, subpop_order)]
```

This function will plot the colors of interest. We encode it as a function since it will be reused a few times.

```
plot_colors_subpops <- function(colors, y = FALSE) {
    # number of individuals
    n <- length(colors)
    # positions to place colors on, as matrix (what 'image' requires)
    if (y) {
        # on y-axis it must be a row matrix going from last to first individual
        x <- rbind( n : 1 )
    } else {
        # on x-axis it must be a column matrix going from first to last individual
        x <- cbind( 1 : n )
    }
    # actually plot the image
    image(x, col = colors, axes = FALSE, useRaster = TRUE)
}
```

Test this plot:

```
# no margins here
par(mar = c(0, 0, 0, 0) + 0.2)
# plot x-axis version (y = FALSE)
plot_colors_subpops( fam$col )
```

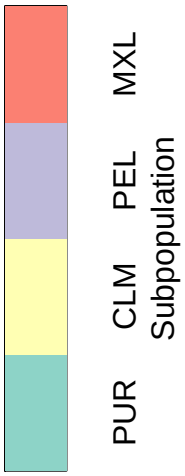
We also need to construct a legend for these colors.

```
legend_color_categories <- function(colors, categories, label, cex_label = 1) {
    # color legend of subpopulations
    x <- 1 : length(colors)
    image(y = x, z = rbind(x), col = colors, xaxt = "n", yaxt = "n")
    axis(4, at = x, labels = categories, tick = FALSE)
    # lastly, add axis label
    mtext(side = 4, label, line = 2, cex = cex_label)
}
```

Test also, using dimensions that make it look tall (it's going to go on the side):

```
# only right margin must be large, for label
par(mar = c(0, 0, 0, 3) + 0.2)
```

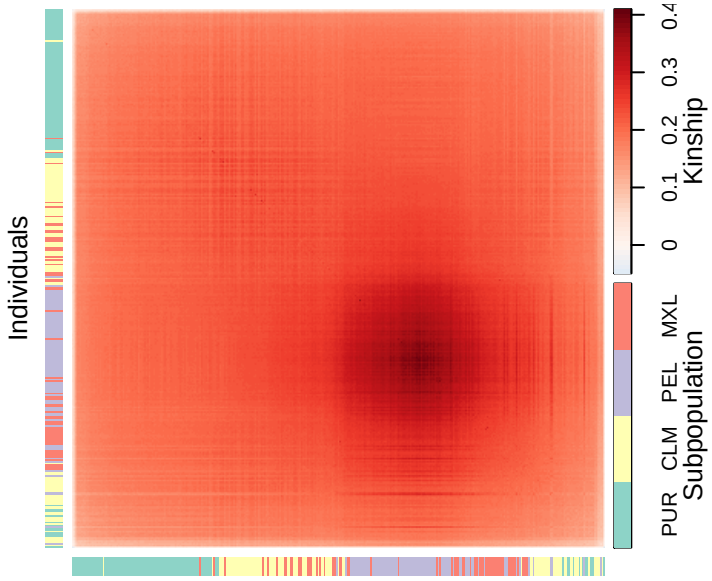
```r
# plot color legend
legend_color_categories(
    colors = colors_subpops,
    categories = subpop_order,
    label = 'Subpopulation'
)
```



Now let's incorporate these colors into the kinship matrix figure. This is more complicated as we need to define a custom layout:

```r
par(oma = c(0, 1.5, 0, 3))
# zero margins here (plus minimal padding)
par(mar = c(0, 0, 0, 0) + 0.2)
# custom layout
layout(
    # Panel numbers are matched in comments below.
    # Main figure (first two rows) is split in two
    # so each color key takes up half of this figure by height.
    rbind(
        c(3, 1, 2),
        c(3, 1, 5),
        c(0, 4, 0)
    ),
    widths = c(0.05, 1, 0.05),
    heights = c(0.5, 0.5, 0.05)
)
# reduce some labels
cex_labels <- 0.8
# now plot!
# panel 1 is kinship heatmap
# panel 2 is kinship color key/legend
plot_popkin(
    kinship = inbr_diag(kinship),
    layout_add = FALSE,
    leg_cex = cex_labels, # shrink to match everything else
    ylab = ''
)
# panel 3: plot y-axis first
plot_colors_subpops(fam$col, y = TRUE)
```

```
# outer label goes next to y-axis
mtext('Individuals', side = 2, line = 0.5, xpd = NA, cex = cex_labels)
# panel 4: then x-axis
plot_colors_subpops(fam$col)
# panel 5: color key for subpopulations
legend_color_categories(
    colors = colors_subpops,
    categories = subpop_order,
    label = 'Subpopulation',
    cex_label = cex_labels
)
```



Admixture analysis

Let's load the admixture proportion estimates we precomputed using the `admixture` software:

```
admixture_proportions <- read_table(
    'data/hispanics-admix-panels.3.Q',
    col_names = c('A1', 'A2', 'A3'),
    col_types = 'ddd'
)
```

The `admixture_proportions` is a numeric matrix, where each row is one individual, and each of the three columns is one of the inferred ancestries. Note that every row sums to one (up to some the 6th significant figure):

```
range( rowSums( admixture_proportions ) )
```

```
## [1] 0.999999 1.000001
```

The above column names "A1, A2, A3" are temporary, as we do not yet know which ancestry corresponds to which column. Let's load the individual annotations to make sense of this data and to map it to our kinship matrix:

```
fam_admix <- read_fam('data/hispanics-admix-panels')
```

```
## Reading: data/hispanics-admix-panels.fam
```

This `fam_admix` table is not the same as the earlier `fam`; the admixture analysis has additional individuals from some reference panels that serve as proxies for the unadmixed source subpopulations:

```
# number of individuals per subpopulation in kinship matrix:
table( fam$fam )
```

```
##
## CLM MXL PEL PUR
##  94  64  85 104
```

```
# number of individuals per subpopulation in admixture analysis:
table( fam_admix$fam )
```

```
##
## CHB CLM IBS MXL PEL PUR YRI
## 103  94 107  64  85 104 108
```

Let's merge the individual IDs and the subpopulation labels with the admixture proportions matrix, so that this information stays linked as we reorder individuals:

```
# individual identifiers
admixture_proportions$id <- fam_admix$id
# subpopulation labels
admixture_proportions$subpop <- fam_admix$fam
```

Let's use the dominant ancestry component in each of the reference subpopulations (YRI: Yoruba; CHB: Chinese; IBS: Spanish) and assign it a continental label (AFR: Sub-Saharan African; AMR: Native American; EUR: European):

```
# codes for TGP subpopulations
reference_subpops <- c('YRI', 'CHB', 'IBS')
# codes for inferred continental-level ancestry
reference_ancestry <- c('AFR', 'AMR', 'EUR')
# best-matching infered ancestries from `admixture`, to calculate
inferred_ancestry <- vector('numeric', k_admix)
# loop through TGP reference subpopulations
for (i in 1 : k_admix) {
    # get subpopulation label
    subpop <- reference_subpops[i]
    # logical indicators of individuals from this subpopulation
    indexes <- admixture_proportions$subpop == subpop
    # extract submatrix for this TGP reference subpopulation
    # only first 3 columns (actual admixture proportions)
    # (the other columns are `id` and `subpop`)
    admixture_proportions_subpops <- admixture_proportions[ indexes, 1:k_admix ]
    # identify the column with the largest weight, store its index
    inferred_ancestry[i] <- which.max( colMeans( admixture_proportions_subpops ) )
}
# if everything went well, this should be a permutation of 1:3
stopifnot(
    sort(inferred_ancestry) == 1 : k_admix
)
# `inferred_ancestry` are indexes, use to reorder the admixture proportions
admixture_proportions[, 1:k_admix ] <- admixture_proportions[, inferred_ancestry ]
# now that ancestry is determined, overwrite the column names with `reference_ancestry`
names(admixture_proportions)[ 1:k_admix ] <- reference_ancestry
```

Now that ancestry has been determined, let's create a plot that verifies the assignments.

```
# order to group TGP subpopulations
# this is the desired order
subpop_order_admix <- c(subpop_order, reference_subpops)
# indexes to reorder individuals
indexes <- order(match(admixture_proportions$subpop, subpop_order_admix))
# reorder rows of `admixture_proportions` only
admixture_proportions <- admixture_proportions[indexes, ]
# adjust margins
par(mar = c(2, 3, 0, 0) + 0.2)
# main admixture plot
x <- barplot(
    t(admixture_proportions[, 1:k_admix]),
    col = colors_admix,
    xlab = "",
    ylab = "",
    border = NA,
    main = '',
    xaxt = 'n',
    xaxs = 'i',
    space = 0
)
# axis labels
mtext('Individuals', side = 1, line = 1)
mtext('Ancestry proportion', side = 2, line = 2)
# add subpopulation labels and lines using a popkin internal function
popkin:::print_labels(
             labs = admixture_proportions$subpop,
             x = x,
             doMat = FALSE
         )
# add basic color legend
legend(
    'right',
    legend = rev(reference_ancestry),
    fill = rev(colors_admix),
    cex = 0.7,
    bg = 'white'
)
```

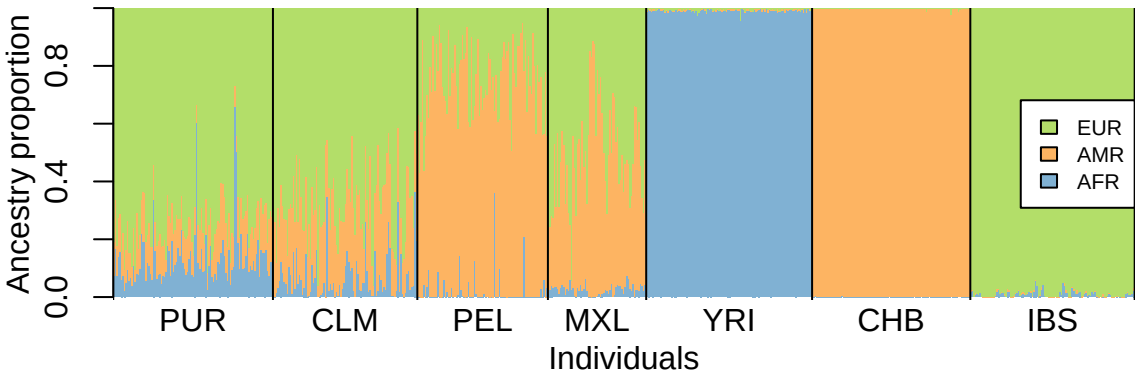


Figure combining kinship and admixture data

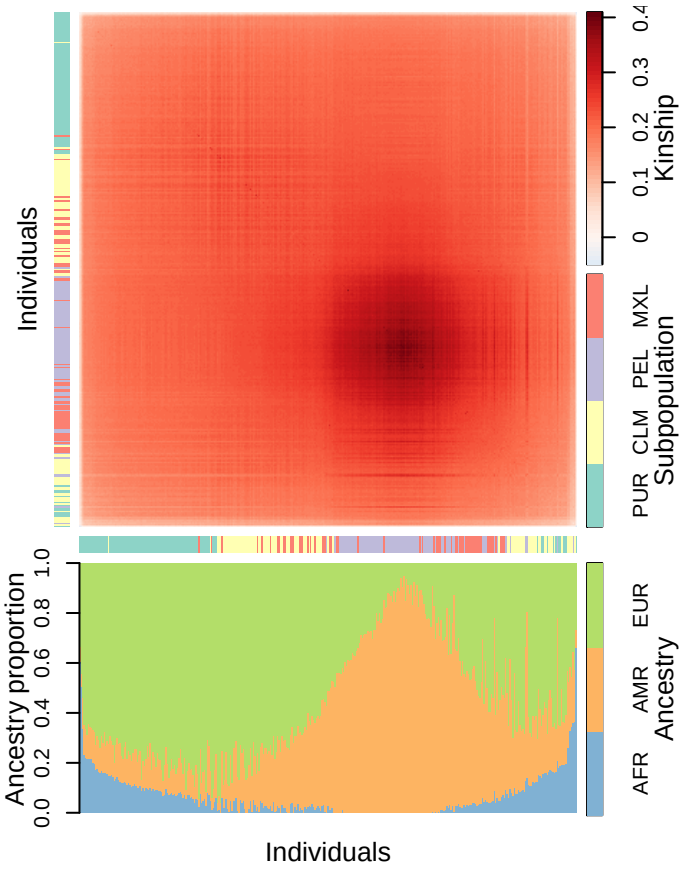
Now we have all the pieces to create a figure combining the kinship matrix and the inferred admixture proportions. First we remove all individuals from the reference panels:

```
# identify where the individuals in the kinship matrix
# (via its FAM table of individual annotations)
# are in the admixture proportions matrix.
# Individuals missing in FAM will be removed
indexes <- match(fam$id, admixture_proportions$id)
# filter and reorder individuals
admixture_proportions <- admixture_proportions[indexes, ]
# sanity check, make sure IDs match
stopifnot(
    admixture_proportions$id == fam$id
)
# now remove ‘id‘ and ‘subpop‘ annotations,
# leaving only the numeric admixture proportions
admixture_proportions <- admixture_proportions[, 1 : k_admix ]
```

Now we plot it. The layout becomes even more complex!

```
par(oma = c(2, 3, 0, 3))
# zero margins here (plus minimal padding)
par(mar = c(0, 0, 0, 0) + 0.2)
# custom layout
layout(
    # Panel numbers are matched in comments below.
    # Main figure (first two rows) is split in two
    # so each color key takes up half of this figure by height.
    rbind(
        c(3, 1, 2),
        c(3, 1, 5),
        c(0, 4, 0),
        c(0, 6, 7)
    ),
    widths = c(0.05, 1, 0.05),
    heights = c(0.5, 0.5, 0.05, 0.5)
)
# reduce some labels
cex_labels <- 0.8
# now plot!
# panel 1 is kinship heatmap
# panel 2 is kinship color key/legend
plot_popkin(
    kinship = inbr_diag(kinship),
    layout_add = FALSE,
    leg_cex = cex_labels, # shrink to match everything else
    ylab = ''
)
# panel 3: plot y-axis first
plot_colors_subpops(fam$col, y = TRUE)
# outer label goes next to y-axis
mtext('Individuals', side = 2, line = 0.5, xpd = NA, cex = cex_labels)
# panel 4: then x-axis
```

```
plot_colors_subpops(fam$col)
# panel 5: color key for subpopulations
legend_color_categories(
    colors = colors_subpops,
    categories = subpop_order,
    label = 'Subpopulation',
    cex_label = cex_labels
)
# panel 6: admixture proportions
barplot(
    t(admixture_proportions),
    col = colors_admix,
    xlab = "",
    ylab = "",
    border = NA,
    main = '',
    xaxt = 'n',
    xaxs = 'i',
    space = 0
)
# axis labels
mtext('Individuals', side = 1, line = 1, cex = cex_labels)
mtext('Ancestry proportion', side = 2, line = 2, cex = cex_labels)
# panel 7: color key for admixture analysis
legend_color_categories(
    colors = colors_admix,
    categories = reference_ancestry,
    label = 'Ancestry',
    cex_label = cex_labels
)
```



This figure demonstrates that ordering individuals using the kinship matrix only results in a smooth ordering in admixture proportions too! These admixture proportions explain the observed kinship matrix very well, in that individuals with greater Native American ancestry share greater kinship compared to individuals with greater European ancestry, while individuals with the greatest Sub-Saharan African ancestry share the lowest kinship values. This is all consistent with the well-known serial founder effects that humans experienced in their migrations from Africa to the rest of the world.

Standard Kinship estimate

We use the `kinship_std` function from the `popkinsuppl` package, which also works on `BEDMatrix` objects and controls memory just like `popkin`. The `mean_of_ratios = TRUE` option ensures that we use the most common formulation of this estimator.

```
kinship_standard <- kinship_std(X, mean_of_ratios = TRUE)
```

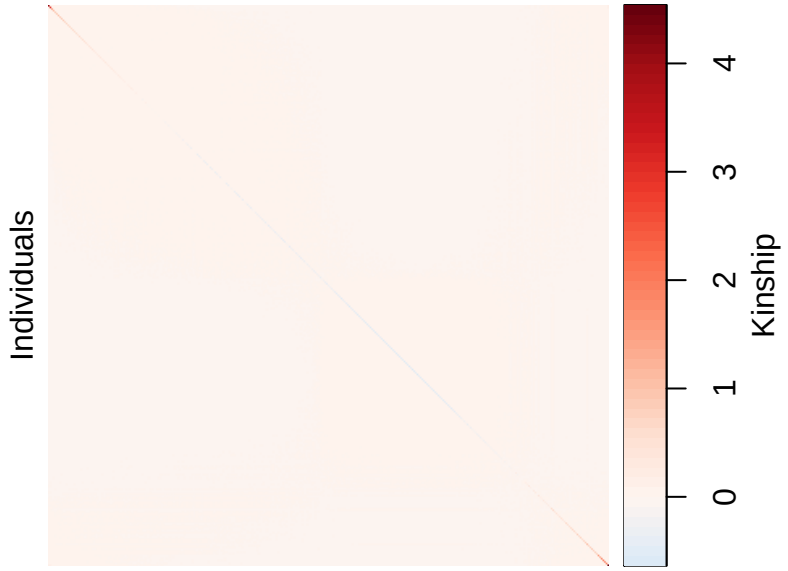
Next we reorder individuals to match the kinship matrix estimated earlier with `popkin` and ordered using `seriation`. Both kinship matrices have individual IDs stored in the row and column names, which we use for sorting.

```
# indexes to reorder individuals
indexes <- match(rownames(kinship), rownames(kinship_standard))
# apply reordering to both dimensions
kinship_standard <- kinship_standard[ indexes, indexes ]
# sanity check to make sure it was done correctly
stopifnot( rownames(kinship_standard) == rownames(kinship) )
# transform diagonal to have inbreeding values
```

```
# make a copy of the matrix, so we retain the original
kinship_standard_copy <- inbr_diag(kinship_standard)
```

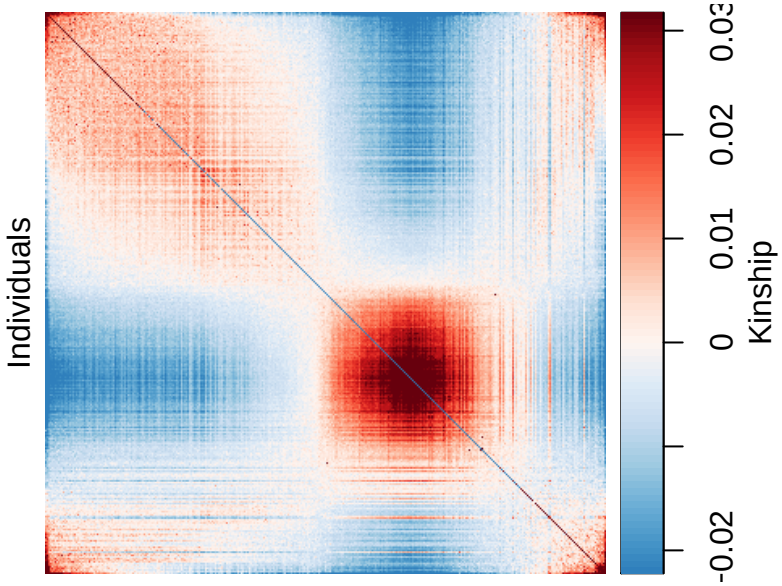
Now we plot this kinship matrix estimate.

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



The biases along the diagonal are so extreme that they make everything else look too small. Note in the color key above that the most extreme values go higher than 4! Since the diagonal is the minority of the data, we cap using quantiles of the overall data.

```
# cap the estimate to this percentile to maximize dynamic range
alpha <- 0.01
# high cap
cap_hi <- quantile(kinship_standard_copy, probs = 1 - alpha)
# low cap
cap_lo <- quantile(kinship_standard_copy, probs = alpha)
# apply caps to the copy only
kinship_standard_copy[kinship_standard_copy > cap_hi] <- cap_hi
kinship_standard_copy[kinship_standard_copy < cap_lo] <- cap_lo
# plot again
par(oma = c(0, 1.5, 0, 3))
par(mar = c(0, 0, 0, 0) + 0.2)
plot_popkin(kinship_standard_copy)
```



The many negative values, distortions along the diagonal and around the edges are clearly visible here. This estimate does not agree with the African Origins model of serial founder effects, which would assign individuals with greater Sub-Saharan African ancestry the lowest kinship, unlike this estimate.

Comparison to existing FST estimators

Here we compare the FST estimated by `popkin` to that of the Weir-Cockerham estimator (Weir and Cockerham 1984) and the "Hudson" estimator (Bhatia *et al.* 2013) generalized to more than two subpopulations in our work (Ochoa and Storey 2016).

The genotype matrix `X` is in the original order for individuals, but the individual annotations table `fam` was reordered (along with the kinship matrix) using seriation. Below we use `fam_orig` instead, which is the same as `fam` but in the original order.

```
# make sure that X and fam_orig agree
stopifnot(
    fam_orig$id == rownames(X)
)

# compute and store the values
Fst_popkin <- fst( kinship )
Fst_wc <- fst_wc( X, labs = fam_orig$fam )$fst
Fst_hudson_k <- fst_hudson_k( X, labs = fam_orig$fam )$fst

# inspect them
Fst_popkin
```

```
## [1] 0.2326449
```

```
Fst_wc
```

```
## [1] 0.02599992
```

```
Fst_hudson_k
```

```
## [1] 0.02507689
```

You can above below that our FST estimate is large, and matches the overall FST in Human Origins (see its vignette).

The WC and Hudson K estimators give much smaller values, which are not coherent with the genetic diversity of Hispanics due to admixture from three continental subpopulations. The labels both of these methods used are the subpopulation labels (PUR, CLM, PEL, MXL), which identify the location of the individuals but does not correspond well with genetic ancestry. It is easy to see, given our earlier kinship and admixture analysis, that these labels do not yield independently-evolving subpopulations (as is required for WC and Hudson K to be correct). Moreover, the earlier kinship and admixture analysis further show that there are no independent subpopulations, so this data is not suitable for FST estimation using the existing approaches. In other words, it is not simply that the labels used were not appropriate, but rather no appropriate labels exist that can possibly yield independent subpopulations in this data.