The 'poputils' Manual

Version poputils,v 0.4 2017/04/12 13:22:58 scott Exp scott $

Scott Hazelhurst

Table of Contents

1 runpca.sh

This is a very simple wrapper for *smartpca.perl* and takes the name of the plink file (without the .bed) suffix. For example

```
runpca.sh BYML
```

If all goes well, the output can be used for *evec2gp*. You can pass extra arguments that the *smartpca.perl* program expects by enclosing them in double quotes:

```
runpca.sh BYML "k=5"
```

2 evec2gp.py

2.1 Purpose

This program can be used to draw the population structure from a principal component analysis. It is a Python program that takes the output of *smartpca* (or something that produces a file of the same structure) and produces a gnuplot program that can be used to draw the structure. The Eigenstrat *proteig* script also does this, but it does it in a different way and doesn't have all the features.

The input file required is the *.pca.evec* file produced by the *smartpca* program. This contains some comment lines starting with a hash (#) and data lines, each of which consists of the same number of fields. The first field is the label of the individual, the last field is some descriptive label and the intermediate fields are the eigenvector of the individual

To actually produce the picture, the gnuplot program is required. This has been tested using gnuplot 4.2 and 4.6 only. It may not work with earlier version of gnuplot.

2.2 Basic usage

We assume in this chapter that the input data is in a file `pop.pca.evec`. The most basic usage is

```
python evec2gp.py pop.pca.evec
```

This will produce a gnuplot file called *pop.gp* and an auxiliary data file called *pop.xdat*. To actually produce the output, run gnuplot

```
gnuplot pop.gp
```

This will produce a picture on the screen showing the first two principal components. **In interactive use, the program is stopped by clicking on the picture with the mouse**

In the simplest mode, there is no differentiation of the population. If you would like to differentiate the population based upon the label in the input data file (i.e., the last column in the data file) use the `--data-has-phe` option.

```
python evec2gp.py --data-has-phe pop.pca.evec
```

If the label is in a separate file, use the `--phe` option. By default, the label is found in column 3 of that file (where we index columns from 1).

```
python evec2gp.py --phe data.txt pop.pca.evec
```

See the description below of the format of the file and how to vary which column should be used.

By default, the gnuplot program will produce the picture on the screen. If you want `gnuplot` (not *evec2gp*) to put its output in a file, you should (a) choose a suitable output form of the picture and (b) give the file name. Use (a) the `--gp-term` and `--gp-output` options.

```
python evec2gp.py --gp-term postscript --gp-output pop.ps  pop.pca.evec
```

Details are shown below. This program is written assuming that the underlying genotype data is stored in plink format. Note there is a differenced in the way plink and smartpca represent invididuals' IDs. The plink program uses a family ID and individual ID as a pair – in the various plink files these are separated by a space. On the other hand, `smartpca` concatenates these two, separating them with a colon.

2.3 Overview of usage

```
python ~/research/gwas/scripts/evec2gp.py -h
```

```
usage: evec2gp.py [-h] [--data-has-phe] [--phe PHE] [--phe-col PHE_COL]                [-
                  [--gp-font GP_FONT] [--gp-size GP_SIZE]
                  [--group-label GROUP_LABEL] [--add-all-labels]
                  [--noput_dots] [--pc PCS] [--epstopdf] [-3] [--perspective]
                  [--eigen_ratio]
                  N

Generate Gnuplot code from evec and related files

positional arguments:
  N                     evec file

optional arguments:
  -h, --help            show this help message and exit
  --data-has-phe        evec file has phenotype (default empty)
  --phe PHE             phenotype file (default empty)
  --phe-col PHE_COL     column in phenotype file (default 2)
  --outbase FNAME       base of output files created (default evec-file-base.)
  --gp-output GP_OUTPUT
                        where gnuplot's output should go (default none)
  --gp-term GP_TERM     gnuplot terminal (default x11)
  --gp-font GP_FONT     gnuplot font (default Helvetica)
  --gp-size GP_SIZE     gnuplot font size (default Helvetica)
  --group-label GROUP_LABEL
                        which groups to label (default none)
  --add-all-labels      add text labels (default False)
  --noput_dots          draw dots (default False)
  --pc PCS              principal components (default 1:2 or 1:2:3)
  --epstopdf            gp program runs epstopdf (default False)
  -3                    plot in 3d (default False)
```

```
  --perspective         produce perspectives for 3D images (default No)
  --eigen_ratio         use eigenvalue for aspect ration for 2D (default  No)
```

2.4 Detailed description

2.4.1 Using phenotype

A key point of this program to draw a picture that shows different groups of indivduals differently. For example, we may wish to show cases or controls in different colours, or individuals from different groups with different labels. The following options control how these groups are defined

2.4.1.1 `--data-has-phe`

This option takes no argument and assumes that the group labels are to be found in data file itself. This assumes two things: (a) the first column of the data contains the id in the format `FID:IID` (the ID of the individual has a colon in it).

2.4.1.2 `--phe FNAME`

This option takes as an argument the name of a file that contains the labels. The file should be in the format of a plink phenotype file – the first two columns are the plink FID and IID. By default, column 3 (indexing from 1) contains the label of the individual.

2.4.1.3 `--phe-col INT`

Use this in conjunction with `--phe` if you wish to choose a different column as the label.

2.4.2 Gnuplot options

2.4.2.1 `--outbase FNAME`: The name of the gnuplot program

By default if you give as input a file *data.evec.pca*, this program will produce two outputs a file *data.gp* which is the gnuplot program and *data.xdat* which is an auxiliary data file needed by *data.gp*. As you may want to analyse the same data in different ways, it's useful to change the output name

```
python evec2gp.py --outputbase result data
```

Note that if you do **NOT** specify the base, your input file *must* have an *pca.evec* suffix. If you do spectify the base then you can choose whatever suffix you want (or not have a suffix).

2.4.2.2 `--gp-output GP_OUTPUT`

This option control how gnuplot produces its output. By default when you run the resulting gnuplot program, the picture is drawn interactively. With this option, output is sent to a file (typically for inclusion in a text document). While not mandatory, typically you should use the this option with the `--gp-term` option.

2.4.2.3 `--gp-term GP_TERM`

Gnuplot is capable of producing output in a wide-variety of output formats. You can specify any format that your gnuplot version/computer system supports, but there is particular

support for PostScript, Aqua, X11 and PDF, described below. In Gnuplot this format is chosen by setting the *terminal* type.

Note that different *Gnuplot* terminal styles use different colours and labels for plotting – so if your PDF, GIF and X11 plots may look quite different. This is a bug/feature of `gnuplot` rather than of this program and is to some extent caused by the fact that different media have different properties. Some colours may display very well on a screen but be invisible on a printed page (or vice-versa – see the discussion on Gnuplot options below). For non-interactive formats you should use `--gp-output` as well.

- x11
  For interactive use – this is the default format. See the discussion below on 3D plots.
- aqua For interactive use – available for Apple (rather use X11 but you can use this)
- postscript
  This is the recommended format for pictures that are to be set in a document that is printed. This is for non-interactive use.
- pdf
  For non-interactive use to produce PDF files. However, the postscript terminal is generally much better supported so I recommend that you use the *postscript* option and convert to PDF if necessary.
- jpeg
  For non-interactive use.
- png
  For non-interactive use.

2.4.2.4 `--gp-font FONTNAME`

Set the font that `gnuplot` uses. By default this is Helvetica.

2.4.2.5 `--gp-size SIZE`

Set the size of the font – by default this is 16.

2.4.2.6 `--epstopdf`

This should be used in conjunction with `--gp-term postscript`. When the `gnuplot` program runs, the PostScript terminal is used and then the *epstopdf* utility is automatically run to convert to PDF. Although more indirect than doing this directly, this option is better supported (the gnuplot PDF option is not available on any systems) and the quality is better. Of course this requires that the *epstopdf* program is available on your system.

2.4.2.7 `--put-dots`

By default, gnuplot will plot "points" which are relatively big. If you have a very big data set use "dots" rather than "points". This is probably only worth using if you have many thousands of participants in your data. It is also useful to use with the `--add-all-labels` and `--group-label` options.

2.4.3 Choosing and Displaying Principal Components

By default the first two principal components are chosen, you can specify the PCs to be used. You can also plot 3 PCs against each other using gnuplot's 3D plotting

2.4.3.1 `--pc PCS`

This can be used to specify the PCs. By default the value is 1:2 (if 2D plotting done, or 1:2:3 if 3D plotting is done. For example, plot the second and fourth PCs.

```
python evec2gp.py --pc 2:4 data.pca.evec
```

If doing 3D plotting (see below)

```
python evec2gp.py --pc 2:3:4 data.pca.evec
```

2.4.3.2 `-3`

Plot in 3D using gnplot's *splot* facility. Note that if X11 is the terminal type you can use the mouse or keyboard to navigate around. This is very useful for visualing the what the population structure is. By default the first three PCs are shown. You can set using `--pc`.

Note that if you are doing this interactively, you may first want to run gnuplot and then load the file `load "data.gp"` so that you can interact more effectively.

2.4.3.3 `--perspective`

Showing perspctives.

The 3D view is a very powerful way of visualising 3D plots interactively, but for non-interactive work the rendering of a 3D plot by a projecting on to a 2D plane may not be useful if the default perspective projects the 3D image in such a way that some of the data is precluded. If the `--perspective` option (which takes no arguments) is chosen, a set of different perspectives of the 3D image is shown.

If an interactive terminal is chosen, then the user must click on the mouse button to show each successive perspective.

If output is sent to a file, then the default perspective is sent to the named file and the other perspectives are sent into files with names with slight variations (the perspectives' angles are included in the file name.

For example:

```
python ~/research/gwas/scripts/evec2gp.py \
       --gp-term postscript --gp-ouptut picture.eps --epstopdf \
       --outbase sample \
       -3 --perspective  \
       --data-has-phe data.pca.evec
```

creates two files *sample.gp* and *sample.xdat*. Then when we run gnuplot

```
gnuplot sample.gp
```

we will get five pairs of files (PostScript and PDF) of different perspectives.

```
cpicture.eps
picture.pdf
picture-120-120.eps
picture-120-120.pdf
```

```
picture-150-60.eps
picture-150-60.pdf
picture-330-300.eps
picture-330-300.pdf
picture-60-30.eps
picture-60-30.pdf
```

2.4.3.4 Aspect ratio

This option is available for 2D plots only. By default, `gnuplot` produces a squarish picture. This may be misleading because the different eigenvectors correspond to different eigenvalues which convey the relative importance of the eigenvectors. If the `--eigen-ratio` option is used (no arguments) the program will expect that the eigenvalues of the PCs are found as a comment line in the first line of the input file (which is the normal behaviour of `smartpca`). The picture of the PCs will be scaled according to the size of the the PCs.

2.4.4 Labels

Sometimes we may wish to label particular points in the graph with the label of the corresponding individual.

2.4.4.1 `--add-all-labels`

Each individual will be labelled. This is normally very cluttered.

2.4.4.2 `--group-label GROUPS`

Only the individuals in the groups labelled (as given by the phe file) will be shown.

```
evec2gp.py  --group-label CEU,YRI,LWK --phe population.phe data.pca.evec
```

This will display only the individuals in these groups.

It is best to use the `--with-dots` option.

2.4.4.3 `--indiv-label fname`

The file should contain a list of individuals who should be labelled. Note that if the individual is specified as a family ID, individual ID separated by a space, then the space is replaced with a colon.

2.4.4.4 `--indiv-label asis:list-of-individs`

If the characters *asis:* is used as the first five characters of this argument, the the rest is assumed to be the list of individuals to be labelled. The individuals are separated by commas.

2.4.5 Using Gnuplot

It is difficult to produce a gnuplot file that works well in all circumstances because the picture is so dependant on the data. For example, the clarity of the picture may depend on the juxtoposition of the colours chosen. Clusters coloured red and orange may be well distinguished if compact and relatively close together where the eye can clearly distinguish them. However, when far apart or if not clearly separate they may be impossible to use.

Thus manual editing of the Gnuplot file is inevitable. You may wish to change

- Where the key is (using `set key`). Options are
  - set key top right
  - set key bottom left
  - variations on this theme
  - set key off
- Number of tics on an axis

  You will find this given on lines like –

  ```
  set ztics 0.04000
  ```

  The number is the increment; increase it if you want fewer, decrease it if you want more.
- Spacing between the axis and the label.
- Fonts and font sizes – labels and keys can be done differently

3 popify.py

Some of the pictures *ploteig* creates and tests that *eigenstrat* does (e.g., work out ANOVA for different groups against PCs) require that the group be in the *fam* file. This could be the standard plink affection status (1=control, 2=case) or a population label. However, in many cases, *fam* files do not have a phenotype (e.g., a pure population structure study) or the user migh wan to change label (e.g., do the tests for Case/Control and/or for several pheonotypes).

popifyfam takes a fam file and phentoype file as input and produces a new fam file using one of the columns of the phenotype column as the phenotype to be used. A phenotype file is a file that has as its first two columns the family and individual ID, followed by one or several phenotype columns. Note that each individual in the fam file *must* be in the phenotype file; however, the individuals don't have to appear in the same order and there can be individuals in the phenotype file that are ot in the fam file.

This is the general form for calling *popify*

```
popifyfam.py -h

popifyfam.py  --popfname PHENOFILE --popcol N --output NEWFAM  OLDFAM
```

The `--output NEWFAM` is optional – output will be sent to standard output. For example, we might say

```
popifyfam.py  --popfname group2.phe --popcol 5 --output new.fam  ALL.fam
```

This takes the ALL.fam file, the *group2.phe* phenotype file and a column number. It creates a new output fam file that is the same as the input except that the phenotype values in the original file are replaced by the appropriate phenotype values from the phenotype file.

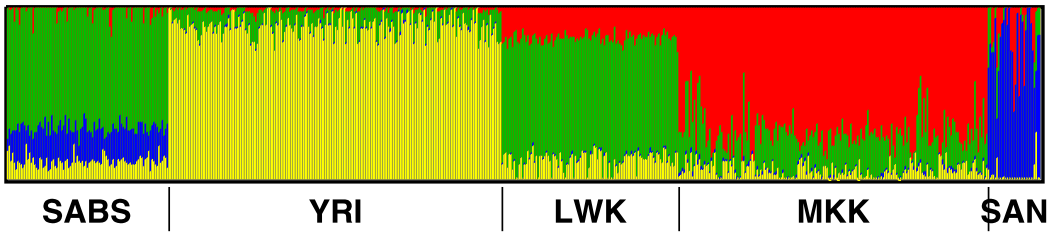
Once the new fam file is created, you can call smartpca.perl – as the fam file does not have the same base name as the other files, you have to call smartpca directrly

```
smartpca.perl  -i ALL.bed -a ALL.bim -b new.fam \
       -p ALL.pca -e ALL.eval -o ALL.pca  \
       -q NO -l ALL.log
```

4 sortfamQwithin.py

This is a useful auxiliary script for structure charts. In an admixed population, individuals within the group are likely to have different proportions. Usually the ordering of the individuals is arbitrary which is likely to lead a very jagged looking structure chart which not only looks ugly but fails to reveal structure. This script will order each population so the similar individuals are close to each other

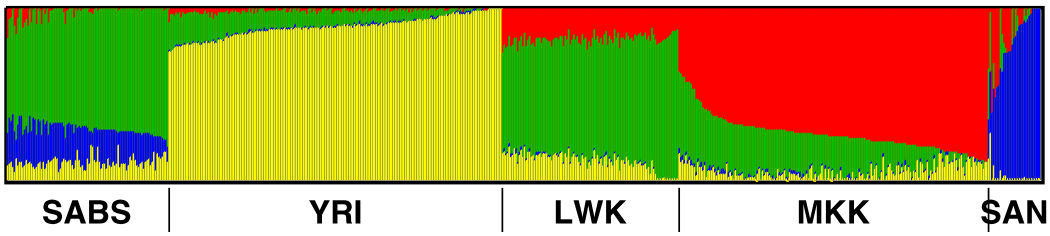
This is a fairly well behaved, real example. As the data came, we got a structure chart like this. As can be seen some of the populations are admixed and so the chart is fairly jagged.




```
python sortfamQwithin.py --poplist SABS,YRI,LWK,MKK,SAN dat.fam dat.4.Q all.phe
```

This creates a new fam file `dat_srt.fam`, and a new Q file `dat_srt.4.Q` which sorts each population separately. In this example, the program will look at all the individuals labelled SABS in the all.phe file, and then sort the fam and Q files by the major "colour" or ancestral composition for the SABS population. Then the same will be done for YRI, LWK and so on in turn. By sort, I mean that that fam and Q files are consistently sorted so that they still correspond with each other.

We now get two new files `dat_srt.fam` and `dat_srt.4.Q` that are consistent with each other. If we draw the new files we get the following, which I think is cleaner and clearer.



We've just seen a typical call, where it is applied on one file. The program takes four compulsory arguments and several optional ones. The compulsory ones are in order a fam file, a Q file as produced by Admixture or CLUMPP outfile, a phenotype file (But only the phenotype in the last column is used), a comma separated list of populations. A new FAM and Q file are created. The program tries to guess intelligent output names based on typical file names produced by CLUMPP and admixture.

4.1 Other features

It also has useful new features that have been added that have nothing to do with sorting because there is common code. Sorry.

- You can summarise Q values by population (e.g., you want to know for K=3, what the average Q values are for all CEU in the data).
- You can filter out populations from a Q and fam file (v error prone to do manually)

4.1.1 Overview

- The `--show-pops-q` option is used to summarise the results by population. It produces a text version of the Q values and a new fam file.
- By default the file names are given the base name of the input FAM file with `_srt` appended. But a new base name can be set using the `--output` option.
- The `--clumpp` option must be given if the input is a CLUMPP-style file.
- The `--debug` option displays the invididuals from the chosen populations and the sorted Q values.
- `--popcol`. By default the rightmost colum of the phenotype file is used as the population but you can specify another. Column numbers are according to Python convention: the leftmost column is 0, the rightmost is -1.
- `--column-for-order`. By default, this script sorts each population by the most important colour for that population. However, the user can specify a column with which to sort all the populations. The `--column-for-order` takes one parameter, which specifies which column in the Q file should be used for sorting the populations. Columns are numbered according to Python conventionL the leftmost column is0, the rightmost is -1
- `--likeness-order`. This is an experimental option which may be removed or changed. This tries to order individuals within a population so that similarly coloured individuals are close to each other. This is slow....
- `--poplist`. This is a comma-separated list of populations that appear in the PHE file. If this is NOT given, the default is that all populations will be sorted
- The `--filter` option allows you to extract the named populations rather than sorting them.

The script can also be used to sort several Q files with a common fam file at the same time. This is often useful for applications where you are doing a stacked structure chart. There are two modes for doing the sorting with different pros and cons. Both are reasonable but it is important to properly describe to your reader what you have done.

In both cases, you need to specify several Q files as the arguments to the script. This is done by givin a space separated list of names or globs. You must either specify `--sort-multi-prettily` or `--sort-multi-consistently` switches.

The first mode (controlled by the switch `--sort-multi-consistently`) is probably the most accurately scientifically. It's easiest explained with an example:

```
sortfamQwithin.py --sort-multi-consistently all-prune-6.outfile
          --clumpp all-prune.fam all.phe SMX,CHS,CLM *outfile
```

Here you are wanting to sort all the *outfile*s in the current directory. The switch `--sort-multi-consistently` takes one argument, one of the Q files you specify in the main list of arguments. You will be sorting the *all-prune-6.outfile* Q file and the fam file. Then all the other outfiles will be sorted according the newly sorted fam file one the basis of the proportions of the *all-prune-6.outfile* file. The advantage of this is that when you show the stacked structure chart, that in each chart you find the same individual at the same horizontal position. The disadvantage is that sorting for K=6 doesn't mean that K=3 will be sorted properly (but that may say something about the stability of the colouring.

The `--sort-multi-prettily` switch does not take an argument. Each Q is sorted according to its proportions and a fam file consistent with it is produced. You can then stack these as appropriate. This is much cleaner and shows structure clearer but may give a misleading view of the stability of the clustering. Also, if you are using a tool like Genesis and want to explore the data you cannot rely on the right person person being the the expected position. For each value of K, the fam file ordering is different – if you have an SMX person at position j for one value of K you can rely on their being an SMX person at position j in all the other fam files, but it is highly likely to be someone else. In summary this produces pretty pictures but it may be misleading so should be described properly.

```
usage: sortfamQwithin.py [-h] [--poplist POPS] [--show-pops-q] [--filter-pops]
                         [--clumpp] [--debug] [--sort-multi-prettily]
                         [--sort-multi-consistently CONSISTENTLY]
                         [--output OUTPUT] [--popcol POPCOL]
                         [--column-for-order ORDERCOL] [--likeness-order]
                         FAM PHE Q [Q ...]

sort fam and Q files

positional arguments:
  FAM                   fam file name
  PHE                   phenotype file name
  Q                     Q file name

optional arguments:
  -h, --help            show this help message and exit
  --poplist POPS        list of populations comma separated (default all)
  --show-pops-q         show pops vals
  --filter-pops         produce new files which are filtered
  --clumpp              Set if a CLUMPP Q file
  --debug               show the chosen pops and Q vals
  --sort-multi-prettily
                        sort multiple Q files sorting for aesthetic value
  --sort-multi-consistently CONSISTENTLY
                        sort multiple Q files consistenly with a fam file (key
                        Q file)
  --output OUTPUT       output name (default is append _new)
  --popcol POPCOL       column in popfile (number from 0) default is -1
                        (rightmost
```

```
  --column-for-order ORDERCOL
                        Use this colour for order
  --likeness-order      sort on closeness to each other
```

Restrictions which I hope may be fixed in the future:

- The program can't automatically detect whether CLUMPP-style or Admixture-style input is given. The default is the former.

4.1.2 Case study

One example of the use of this script came in the SAHGP pilot phase paper. In an admixture study we had about 800 invidividuals over a dozen or so groups. To show the global admixture in a figure in the main paper where space is restricted led to something that is so busy that it is illegible. So we decided to show it as follows:

- For each person in the SAHP study (24) we would show as a person, a column by itself
- For each of other groups, we would show one colum showing the summary of that group.

First, we extract the SAHGP24 from the files

```
for x in `seq 2 11`; do
   sortfamQwithin.py --clumpp --filter --popcol 3 \
   --poplist  COL,XHS,SOT,ZUS,XHD pruned.fam all2.phe pruned-$x.outfile\
   --output   sa$x
done
```

This will give us for example sa-7.outfile which is the clumpp Q estimates for K=7. There is a corresponding fam file.

Now we add the summary for each group to the Q files:

```
for x in `seq 2 12`; do
   sortfamQwithin.py \
   --poplist BEB,LWK,CEU,YRI,SEB2,SWB,SSMP,BagandaAG,CWC,ZuluAG --clumpp \
   --show-pops-q --popcol 3  pruned.fam all2.phe pruned-$x.outfile \
   |  sed -n "s/ *\(.*\): *\(.*\)/\2/p" >> sa-$x.outfile ;
done
```

And the matching fam file (note that there is a common fam file so we only need to pick one arbitrarily as the reprentative)

```
sortfamQwithin.py --poplist BEB,LWK,CEU,YRI,SEB2,SWB,SSMP,BagandaAG,CWC,ZuluAG --clumpp --s
```

5 fams2phe

This is a simple script that takes a list of FAM files and creates a phenotype file. The phenotype given is the name of the fam file. So if there are files YRI.fam SAMPLE.fam. The phenotypes of all individuals in the YRI file have a phenotype YRI; the phentotpe of all individuals in the SAMPLE file are SAMPLE.

Typical usage would be

```
fams2phe YRI.fam sample.fam
```

```
fams2phe *fam
```

All output is to standard output

6 admix.sh

Running admixture or structure is easy. However, in some cases, you may want to run admixture many times. First, you may run admixture for different values of K (the number of groups). Second, for programs like CLUMPP it is recommended you run admixture many times (even up to 100) for every value of K. This is not practical to do manually. There are two scripts that can be used to automate the process. They both have the same functionality but differ in how they work.

To use the scripts, create a directory where you want the output to go. Set the `OUTDIR` bash variable to the name of this directory. Decide how many times you want to run admixture for each value of k and set `NUMRUNS` to that variable. Decide which values of k you want to explore, and set `MINK` and `MAXK` to set the range. Set `INPDIR` to be the directory where the input plink files can be found and `DATAFILE` to be the name of the input file (without the .bed suffix).

When the scripts run, they will create directories called 0, 1,... $NUMRUNS$-1 in which the output of the admixture program for each run is put. This is a good directory structure for the *cdg* program discussed later. Note that if you want to run the scripts on different data files, provided the data files have different names, it works well to use the same output directory for all the input files.

The first script *admix-simple.sh* is a bash script that sequentially runs the *admixture* for each run and each value of k. The second script *admix-torque.sh* is suitable for running on a cluster using Torque/PBS as the computinh platform. It is highly desirable to use cluster since for realistic data sizes and parameters computation may require several hundred CPU hours.

One thing to note is that *admixture* can run in parallel mode using multiple cores. This is set with the `-j` option. The script uses `-j2` but you should change. For *admix-simple.sh* change this to the actual number of cores you have on your computer. For *admix-torque.sh* I suggest you leave it at 2. See the discussion below.

6.1 admix-simple.sh

The only thing that is non-obvious is the explicit use of a random seed for *admixture*. This is probably unnecessary for this script but more neeed for the next script where it could be that the multiple runs of admixture run at the same time. So the setting of the N and S variables in the script just generate a semi-pseudo-random number.

```
DATAFILE="BYLMK-G04-M02"
OUTDIR=/home/scott/sample
INPDIR=/share/novartis/combineddata

echo ${DATAFILE}
NUMRUNS=100
MINK=2
MAXK=8


for T in `seq ${NUMRUNS}`; do
```

```
    for K in `seq ${MINK} ${MAXK}`; do
       N=`date +%N | cut -b1-6 | sed -e "s/^0\+//"`
       S=$((${N}+41217*${K}+17*${T}))

       echo  "K=$K, T=$T, N=$N, S=$S"

       cd $OUTDIR
       mkdir -p ${T}
       cd ${T}
       echo "admixture -s ${S} ${INPDIR}/${DATAFILE}.bed ${K} -j2 "
       admixture -s ${S} ${INPDIR}/${DATAFILE}.bed ${K} -j2
    done;
done;
```

6.2 admix-torque.sh

The *admix-torque.sh* script is similar, except that rather than using loops, we uses a Torque/PBS feature that allows one Torque job ro spawn multiple jobs. This is the `-t` option: for each value in the specified range, Torque launches the same job, except witha different value of the shell variable `PBS_ARRAYID`. We use this to differentiate between jobs. In our case, we use this value to determine both the value of `K` and which run it is.

To use the script, set the environment variables for the directories and the file name in the script as mentioned above. In addition, you should set the following

- The value of `-t`. Suppose you want to run the script for values of k from $MINK$ to $MAXK$ (inclusive) and for each value of k run the job $NUMRUNS$ times. Then t should go from $MINK * NUMRUMS$ to $MAXK * (NUMRUNS + 1) - 1$. For example if we want to run the script 100 times each k going from 2 to 7 (inclusive), then use `#PBS -t 200-799`.
- The value for *ppn* must match your choice for the `-j` option of *admixture.sh*. It may seem that as we are running possible hundreds or even thousands of jobs (which run in parallel with each other) there is no point in parallelising and individual run. However, there is some benefit in using the parallel option for *admixture* here to reduce memory footprint (running four jobs each using two cores, rather than 8 jobs each using one core requires half the amount of memory). However, using higher values of k may complicate scheduling so I think 2 is a good compromise
- Set the mem value. A value of 1GB is likely to be sufficient. But you are doing some substantial computing here, so experiment. Run admixture on your data and then use top to see how much memory is being used and add a little extra for safety.

```
#!/bin/bash
#PBS -N admixture-novartis-bylmk
#PBS -l nodes=1:ppn=2,walltime=100:00:00,mem=4GB
#PBS -q WitsLong
#PBS -t 200-799
#PBS -o /tmp
#PBS -e /tmp
```

```
# t should go from MINK*NUMRUMS to MAXK*NUMRUNS-1


DATAFILE="BYLMK-G04-M02"
INPDIR=/share/novartis/combineddata
OUTDIR=/home/scott/spain

echo ${DATAFILE}
NUMRUNS=100

K=$((${PBS_ARRAYID}/${NUMRUNS}))
T=$((${PBS_ARRAYID}%${NUMRUNS}))


N=`date +%N | cut -b1-6 | sed -e "s/^0\+//"`
S=$((${N}+41217*${PBS_ARRAYID}))

echo  "K=$K, T=$T, N=$N, S=$S"

cd $OUTDIR
mkdir -p ${T}
cd ${T}
echo "admixture -s ${S} ${INPDIR}/${DATAFILE}.bed ${K} -j2 "
admixture -s ${S} ${INPDIR}/${DATAFILE}.bed ${K} -j2
```

7 cdg

The `cdg` takes as input a set of independent outputs from the `admixture` or like program and produces suitable parameter files for CLUMPP and Distruct. It can also call these programs. The final output is a PDF file with the structure picture.

7.1 Basic call

Here are some simple calls

```
python cdg.py -K 8 --popfname family.phe --par_clumpp sample
```

This produces a parameter file *paramfile* that can be used by the CLUMPP program. This by default looks for any files of the form *sample.8.Q which can be found in the current directory. It is assumed that these have been produced by runs of the `admixture` program, expecting the population to be in 8 clusters. This also assumes that there is a file called *sample.fam* which in `plink` *fam* format and matches the `admixture` output files.

The file *family.phe* is in `plink` phenotype format: the first two columns are family and individual ID and the third column is some identifying information (typically a population)

```
XAB 0 CEU
XAC 0 CEU
XDD 1 YRI
```

Note that the `-K` and `--popfname` parameters are mandatory.

7.2 A typical run

```
python cdg.py -K 8  \
       --glob "[0-9]/" \
       --popfname ../common.phe --popcol 5  \
       --par_clumpp --doclumpp --par_distruct --dodistruct  \
       --order_pops  YRI,LWK,MKK,SANDAWE,HADZA,B2T,SAN \
       --outputps cluster8  \
       common6
```

This says:

- We are clustering into 8 groups
- The input base name is *common6*
- We expect the input to be in directories 0, 1, 2, ..., 9 in the current directory.
- The file common.phe contains the "known" population of each individual. Column 5 contains the population (the default is 3).
- Produce the clumpp parameter file, run clumpp, produce the distruct parameter file, run distruct
- In the distruct output, show the populations in the order given.
- Put the output eps and pdf in cluster8.eps and cluster8.pdf

7.3 The full list

```
usage: cdg.py [-h] -K K [--glob GLOB] [--popfname POPFNAME] [--popcol POPCOL]
              [--popdescr POPDESCR] [--output OUTPUT] [--par_clumpp]
              [--order_pops ORDERPOPS] [--par_distruct]
              [--clumpp_paramfile CLUMPP_PARAMFILE]
              [--distruct_paramfile DISTRUCT_PARAMFILE] [--outputps OUTPUTPS]
              [--doclumpp] [--guesspops] [--dodistruct]
              S

create clumpp paramfile.

positional arguments:
  S                     base file name

optional arguments:
  -h, --help            show this help message and exit
  -K K                  K -- number of clusters
  --glob GLOB           glob
  --popfname POPFNAME   name of pop file in phe format
  --popcol POPCOL       column in popfile (number from 0) default is 3
  --popdescr POPDESCR   (output) file name of num pop
  --par_clumpp          Produce clumpp param file
  --par_distruct        Produce distruct param file
  --doclumpp            run clumpp
  --dodistruct          run distruct
  --clumpp_paramfile CLUMPP_PARAMFILE
                        name of clumpp parameter file (output default is
                        clumpp_paramfile)
  --distruct_paramfile DISTRUCT_PARAMFILE
                        name of distruct parameter file (distruct param:
                        default is dfawparamfile)
  --order_pops ORDERPOPS
                        Order of populations in distruct output
  --output OUTPUT       output name (default is basename)
  --outputps OUTPUTPS   name of output postscript file
  --guesspops           guess matching between pops
```

- `--glob` should be a standard Unix-style glob. You probably need to quote it in your call to prevent the shell from interpreting it.
- `--popfname` is the name of the population file. There is no default value – if you don't use it, then there is no population structure so it probably not useful not to specify it.
- `--popcol` is 3 by default.
- `--popdescr` names a file that lists your (known) populations with an integer number that *distruct* uses. By default this will be named *S.names* where S is the base name you use. This is probably good enough in most cases unless you want to prevent renaming an existing file. Note that *cdg* creates this file.

- `--par_clumpp`, `--par_distruct`. These create the parameter files for clumpp and distruct. By default they files are named *paramfile* and *drawparams* which is what the default names are for the programs. You may want to edit the parameter files to do things that cdg can't do. In this case, I suggest explicitly naming the parameter files to reduce the chance of inadvertently over-wrting an existing file. A good protocol would be to run the program once just creating the parameter files, editing the paramter files and then not using the `--par_clumpp` and `--par_distruct` files.
- `--clumpp_paramfile` and `--distruct_paramfile` These name the parameter files (as explained above). There are t#!/usr/bin/perl $from = "029134"; # 027913 $to = "029509"; $job = "075"; $mac = "pc33"; $sub = "-"; ‘ssh $mac /home/students/coms200f/expers/wcd -d /home/students/coms200f/expers/clt/clt-jobsub$from-$to-$mac -R /home/students/coms200f/expers/H $from $to > /dev/null < /home/students/coms200f/expers/seqlens &‘; wo reasons you might want to do this. The first is explained in the previous point. The second is if you are running the program in parallel with other parameters changing and so do not want files clashing.
- `--order_pops`. This is a comma-separated list of populations that will be used to determine the order of the (known) populations in the data file.
- `--output OUTPUT`.

  This is used as the prefix for any file names the program produces. The default is to use the input file name as the prefix, which makes sense in many situations. However, if you are running the program with the same input file and multiple K values then you should explicitly give the output name (a good trick is to use K in the name)
- `--outputps OUTPUTPS`

  The name of output postscript file – also sensible defaults are chosen but if running multiple times, better to explicitly name
- `--guesspops`

  This is an auxiliary option and should be used after clumpping and possibly before distructing. It cross references the known populations and the K groups that you have clustered into and gives a finger print for each known population. You can use this to help colour final result. Distruct takes as input a file with a .perm suffix which contains on each line the unknown population number (in the range 1 to K) and a colour. Particularly if you are producing several structure plots you may wish to manually specify colours for consistency. For example an output using `--guesspops` might be:

  ```
  Group SABS (12) fingerprint 3 [0.933874],  1[0.032199]

  Group YRI (5) fingerprint 1 [0.926137],  2[0.046378]

  Group LWK (7) fingerprint 2 [0.528994],  1[0.274192]

  Group MKK (9) fingerprint 5 [0.580811],  4[0.205266]
  ```

  This says, for example, that LWK (which is population number 7 in your input population file as found in popfile) is an admixture of the unknown populations 2 and 1 (52 and 27 percent respectivey). In other words known population 7 matches unknown

population 2 and so the collur that is used for population 2 will be the colour that dominates the drawing of LWK. So a file with BASENAME.perm (or check your distruct parameter file for the value of INFILE_CLUST_PERM) like

```
1 orange
2 blue
3 yellow
4 green
5 light_purple
```

would have the effect of making the dominant colour of MKK light purple, LWK blue, YRI orange, and SABS yellow.

7.4 Sample script

```
K=5
INP_DIR=/home/scott/spain
DATAFILE=BYLMK-G04-M02


cd ${INP_DIR}


python ~/lib/bin/cdg.py  --glob "[0-9]*/" -K ${K} \
                     --popfname ../all-common.phe \
                     --par_clumpp \
                     --clumpp_paramfile ${DATAFILE}-${K}.param  \
                     --doclumpp \
                     --par_distruct\
                     --distruct_paramfile ${DATAFILE}-${K}.drawparam  \
                     --dodistruct \
                     --output ${DATAFILE}-${K}-out \
                     --outputps ${DATAFILE}-${K}.ps \
                     ${DATAFILE}
```

8 stratexclude.py

8.1 Overview

This program can be used to find outliers in a population structure. It probably makes most sense to use with data produced doing a principal component analysis but it can also be used with admixture data. It has many features, but they are not all tested properly. At the moment you should only use for PLINK type data

```
usage: stratexclude.py [-h] [--phe PHE] [--fam FAM] [--clumpp] [--smartpca]
                       [--out OUT] [--plink] [--euclidean] [--weighted]
                       [--phe-col PHE_COL] --group GROUP [--analyse]
                       [--num-pcs NUMPCS] [--numoutlieriter NUMOUTLIERITER]
                       [--exclude-threshold EXCLUDE_THRESHOLD]
                       FNAME

Identify individuals who are different!

positional arguments:
  FNAME                 Q file

optional arguments:
  -h, --help            show this help message and exit
  --phe PHE             phenotype file (default empty)
  --fam FAM             name of fam file (default None)
  --clumpp              is the file in clumpp format (default None)
  --smartpca            file in smartpca format
  --out OUT             name of output file
  --plink               file in plink
  --euclidean           use Euclidean distance to remove
  --weighted            weight distances with Eigenvalue
  --phe-col PHE_COL     column in phenotype file (default 2)
  --group GROUP         which group
  --analyse             do analysis
  --num-pcs NUMPCS      Number of PCs to use (0=all)
  --numoutlieriter NUMOUTLIERITER
                        Number of iterations of outlier removal
  --exclude-threshold EXCLUDE_THRESHOLD
                        threshold for exclusion, default is 0 -- just show
                        distances
```

8.2 Explanation

Exclusion of outliers is done on a group basis. We are assuming that all the PC data is good, but what may be anomalous is the label of an individual compared to the PC value. For each group of interest, we separately remove outliers.

The default way of removing per group is component-wise For each principal component, we compute the mean position of all members of that group, and the average and standard

deviation of the distance of each member in that component to the mean. All members whose distance to the mean is greater than the stated threshold are removed.

An example run would be

```
python ./stratexclude.py  --group teddy/grizzly,polar --plink  \
   --phe bears.phe --exclude-threshold 3 --num-pcs 2  --numoutlieriter 2 \
   --out bears.exclude bears
```

The key parameters are

- `--group` Which groups should be considered. If this parameter is not given, then all data is considered as part of one group. The groups are are given in a hierachical way: first the slash is used to separate the groups we are interested in; second within each group, we use a comma to separate the labels that make up the group. In our example, above we have decided to analyse grizzly and polar bears as one group, and Teddy bears as another group.
- `--plink` The input data is in PLINK format, so there is a .eigenvec and out .eigenval pair of input files.
- `--phe` The name of the phenotype file. By default, column 3 is the group label, but this can be modified by the `--phe_col` option.
- `--exclude-threshold` This is the threshold to be used to exclude members of the group. The unit of measurement is the number of standard deviations. So in our example, if the distance of a member of the group to the mean position of that group in any PC-coordinate is greater than this value multiplied by the standard deviation, then that member is delelted. YMMV: SMARTPCA uses 6 for the default but typically SMARTPCA is used for the entire data set so a smaller value may be appropriate.
- `--num-pcs`: How many PCs should be used in this. This needs some thinking about depending on your PC structure. SMARTPCA uses 10 as its default. This is likely to be too big.
- `--numoutlieriter`: How many times should we iterate the exclusion. If we just run this exclusion one then some extreme outliers may mask other outliers. SMARTPCA uses 5 as the default, but probably smaller would work because at this stage you would expect the much tighter groups.
- `--out`: the name of the output file where the members to be deleted will be stored. If the input file is in PLINK format, then the output file will be in a format suitable for the PLINK `--remove` option.
- The final argument is the name of the input file. If you have chosen the `--plink` option, then just give the base name (no suffix).
- `--fam` this is generally not needed for PLINK (unless your fam file for some reason does not have the same base name as the eigenvector file).
- Other options are experimental and should not be used at this point.

This porgram is rather noisy when run, but this is by design. My experience is that it is easy for the user to make a silly mistake. Please look at the output so you can see if there are any anomalies

9 plinkmerge.py: PLINK merging

PLINK can merge multiple files in one shot. However, small anomalies can cause this to fail. plinkmerge.py is a script that takes a set of PLINK files and tries to merge them one by one.

It takes care of SNPs that might be duplicated as well as SNPs that must be flipped.

Warning!!!!!!!!!!!!!! This assumes that you have compatible and relatively well-behaved PLINK files in the first place. In particular it makes the assumptions that (1) You have common SNP names (2) A common genome build; and (3) **NB: that there are no ambiguous A/T or C/G SNPs**. This script should only be employed in a pipeline where care has been taken to detect any problems. A typical application is where you have a set of reference files (e.g. separate population files from 1000 Genomes which you know are well behaved and some of your own data which you know is well behaved, and you have applied your mind to the data)

A typical run would be

```
plinkmerge.py --log res.log -d common  CEU YRI LWK
```

This merges the PLINK files with base names CEU, YRI, and LWK and puts the result in common bed,bim,fam file. The log file *res.log* gives some idea of what happened.

In order to facilitate debugging this script is a little greedy with disk space. There will be a ptemp directory that contains files that should be deleted if all goes well.

The script does the following

- First, we check for any duplicate SNPs in each of the file: we create a new "clean" plink file with the duplicates removed.
- We then merge files one by one. Hopefully plink merge works, but often it fails because of SNPs that are reported on the alternate strand. We check for the existence of the missnp file.
- If it exists, we flip those SNPs and try to merge again. If it fails, we hope it's because of a few multi-allelic SNPs. We then remove any SNPs in the missnp file and do a final attempt at a merge
- The log file shoud be examined carefully.

```
usage: plinkmerge.py [-h] [--log LOG] [--log-level LOGLEVEL] -d DEST
                     bases [bases ...]

Merge a set of plink files.

positional arguments:
  bases                 source bases

optional arguments:
  -h, --help            show this help message and exit
  --log LOG             log file (def: pmerge.log)
  --log-level LOGLEVEL  log level (def: INFO)
```

`-d DEST, --dest DEST  output name`

Note that

- The log file may exist and is *appended* to.
- For `--log-level` the following are permitted in increasing levels of verbosity: ERROR, WARN, INFO, DEBUG. INFO is the default.
- The destination files must NOT exist before the script runs; the script will refuse to run if they do

10 pop-workflow.nf

This is a work in progress – a NextFlow workflow to merge plink files. It has the following dependencies

- plink
- strandcheck.py (a program that removes A/T and C/G SNPs)
- plinkmerge.py (see above)

This is a sample run. The *pops* parameter is a comma separated list of the populations you want to merge.

```
nextflow pops-worflow.nf --pops BSO,KS,YRI,LWK --outname combined
```

Other parameters and defaults are

- geno (0.02)
- mind (0.02)
- prune (0.15, r2 cut-off for LD-pruning)
- hwe (not used in current version)

A key data structure in the workflow is a dictionary called *pfile*. This has as a key the population abbreviation, and value the location in the file system where the file can be found. (In the next version, I'll remove this from the workflow and rather have a separate configuration file).

In the current version (may change), we assume that if the base name of the PLINK file is BSO, then the three PLINK input files will be `BSO.bed`, `BSO_bp.bim` and `BSO.fam`. The implication is that the BIM file will have as it's SNP ID the concatenation of chromosome and base position (but we only care that it is unique).

All output goes to the *plinks* directory.

The workflow extracts the SNP IDs from the files, and takes the intersection of these. It creates a file with the common SNP names and also will create a file of bad SNPs that need to removed.

It then creates new PLINK files from the inputs but only taking the common SNPs, and then merges.

11 Copyright

The program copyright details are:

Concept Index

(Index is nonexistent)