ANCESTRYMAP SOFTWARE DOCUMENTATION

What's new in Version 2.0

The main enhancement in this version is the ability to do a fine-mapping run to follow up a peak in a coarse scan. In addition to doing the fine-mapping run one can also run simulations in this mode. A number of small enhancements have been added as well, such as fast check for duplicates, support for PED files and some extra output. A number of small bugs have been fixed as well. Details of the various enhancements are given later in the documentation in Section 7.

1. Overview of admixture mapping

Admixture mapping is a method for localizing disease causing genetic variants that differ in frequency across populations. It is most advantageous to apply this approach to populations that have descended from a recent mix of two ancestral groups that have been geographically isolated for many tens of thousands of years: for example, African Americans have both West African and European American ancestry. The approach assumes that near a disease causing gene there will be enhanced ancestry from the population that has greater risk of getting the disease. Thus if one can calculate the ancestry along the genome for an admixed sample set, one could use that to identify disease causing gene variants. The figure below shows a schematic of how a disease locus would appear in an admixture scan of patients and controls.

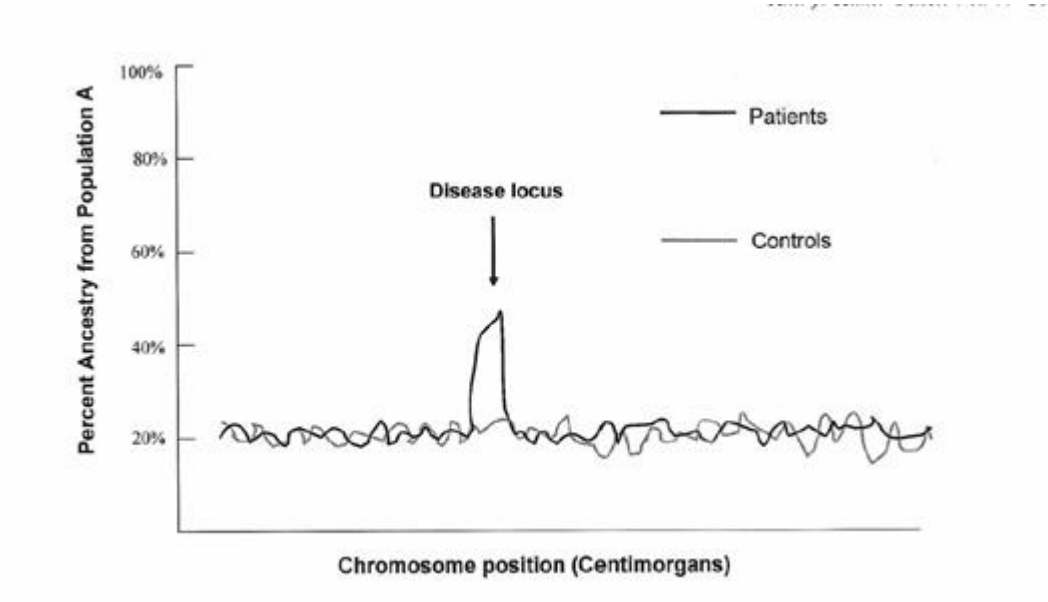

Fig. 1: Schematic of how a disease locus will appear in an admixture scan.

2. Algorithm overview

a) Introduction

In this section we will briefly discuss the algorithm, its limitations, applications and a comparison with other association studies.

Our program estimates the ancestry along the genome of a sample population resulting from recent admixture between two ethnic groups. The program uses data

from individuals genotyped at a set of markers, where the markers chosen are preferably the ones that differ significantly in frequency between the two ancestral populations. Ideally the user should have genotype data both for admixed samples, and for the two parental groups. The approach and algorithm have been described in detail in a paper by Patterson et al.[1] As emphasized in the paper, although controls are not required for screening of disease genes, including them can be useful, in particular for obtaining robust estimates of the marker frequencies in the ancestral populations, which is important for increasing the power of the analysis.

The algorithm calculates a Bayesian-likelihood ratio test to scan for disease association anywhere in the genome. In this calculation, individual ancestry estimates along the genome are averaged across all the individuals to identify genomic regions where there is enhanced ancestry from one of the parental populations, indicating the presence of a disease gene nearby. The algorithm uses a Hidden Markov Model, where the ancestry state is "hidden" and is inferred based on the genotypes, and a model of how data is generated.

The parameters of the model that are of interest are as described in the paper:

- M_i: Average proportion of alleles inherited from population A (where admixture is being considered between populations A and B), for each individual i. This parameter is referred to as θ in the software package.
- M_i^X : Proportion of ancestry A for the X chromosome, for each individual i, referred to as $\theta_{i,X}$ in the software package.
- λ_i: Number of chromosomal exchanges per morgan between ancestral segments of the genome since the mixing event, for each individual i. This can be roughly identified with the number of generations since admixture.
- λ_i^X : Number of generations since admixture on X chromosome, for each individual i.
- p_j^A, p_j^B frequency of the alleles in the parental populations for each marker j.
- τ^A , τ^B : These parameters take into account the uncertainty in the marker allele frequencies due to sampling of a limited number of individuals from populations A and B. These also account for genetic dispersion between the ancestors of a mixed population, and their modern counterparts.

- All of the above parameters are unknown, and are sampled using a Markov Chain Monte Carlo approach, following a "hierarchical Bayesian" framework to perform the calculation. In this approach the initial values are set as follows:
- p_j^A, p_j^B are set to values estimated from genotype data for the parental

populations. For example in the paper by Smith et al[2].we calculated these using genotype data from modern West African and European samples. If the frequency estimates are not provided by the user, the program makes an estimate of these values from the admixed samples.

- M_i: is set for each individual through the use of maximum-likelihood estimates based on treating all SNPs as unlinked.
- λ_i: Is set to 6 for all samples in our case, based on empirical studies.

ANCESTRYMAP starts sampling from the correct conditional distribution after a sufficient number of "burn-in" iterations. By running enough follow-on iterations one can explore the posterior distribution of the various parameters given the data, and obtain a statistic that appropriately takes into account the uncertainty in the unknown parameters. The program minimizes the number of burn-in iterations that are required by using an expectation-maximization algorithm to pick initial values of the parameters that are relatively close to the true values.

This algorithm calculates two separate statistics which can be used to identify disease genes. These are:

1. **Locus-genome statistic**: Compares the percentage of ancestry derived from one of the parental populations at any locus with the average in the genome. This essentially compares for each point in the genome, the likelihood of being a disease locus versus being a locus unrelated to disease. Formally, this is given by for each individual i and each marker j as:

$$L_{ij} = \frac{P(\text{data}|\text{disease})}{P(\text{data}|\text{no disease})} = \frac{\gamma_{i,0}(j) + \gamma_{i,1}(j)\psi_1 + \gamma_{i,2}(j)\psi_2}{\eta_{i,0} + \eta_{i,1}\psi_1 + \eta_{i,2}\psi_2} .$$

where ψ_1 and ψ_2 are the increase in disease risk due to having 1 or 2 population A ancestry alleles, respectively, relative to having no population A ancestry allele. $\gamma_{i,0}(j)$, $\gamma_{i,1}(j)$ and $\gamma_{i,2}(j)$ is the estimated probability for individual i, for having 0,1 or 2 population A alleles at marker j. And, $\eta_{i_{i,0}} = (1 - M_i)^2$, $\eta_{i_{i,1}} = 2\, M_i(1 - M_i)$ and $\eta_{i_{i,2}} = M_i^2$.

To obtain a genome-wide score we will multiply L_{ij} over all the individuals, and then average it at equally spaced points genome-wide. A positive association can be declared if log base 10 (LOD) of the average is greater than 2. For individual markers L_j is referred to as LGS in the output and the genome-wide score is referred to as "genome log factor", for ex.:

>> genome log-factor: 9.028

2. **Case-control statistic**: Compares cases and controls at every point in the genome, looking for differences in the ancestry estimates. This calculates for each individual i and at every locus j in the genome, the difference between their expected number of population A ancestry alleles at a locus and the estimate from data:

$$\mu_i(j) = 2\,M_i - [\,2\,\gamma_{i,2}(j) + \gamma_{i,1}(j)\,]$$

A t-statistic is calculated for a difference of $\mu(j)$ between cases and controls. Both of the above statistics are averaged over all the iterations. The advantages and disadvantages of using one statistic over the other are explained in detail in the paper.

b) Limitations

The program will currently work only for admixture between two populations, and is limited to considering markers that are bi-allelic. Our approach allows admixture mapping to be applied to the X chromosome, which has to be analyzed differently from the autosomes, however currently it does not support the Y chromosome, mitochondrial DNA, or the pseudoautosomal region of the X chromosome. The current implementation has parameters tuned for African American ethnicity for the sample population. If users of the program wish to study different admixed populations, please contact the author. It is important to note that one should exclude samples that have ancestry from only one population ($M_i = 0$ or 1), since they will show no crossover between segments of different ancestry.

c) Applications of the program

- Estimate ancestry along the genome for an individual
- Find disease-causing genetic variants associated with ancestry

d) Comparison to other mapping approaches and guidelines for optimal study design

Admixture mapping has more power to detect genetic variants of weak effect—the type that are likely to be responsible for complex diseases—than linkage mapping, the classic approach of mapping in families that has been so successful for rare, Mendelian disease.

Admixture mapping has a great advantage over linkage mapping because it is a type of association analysis, like whole-genome haplotype mapping or candidate gene analysis, and thus has much more power to detect risk variants of weak effect. However, it differs from other association mapping methods in two important respects, which have a major impact on study design. (1) It requires 100-1000-times fewer markers to carry out a whole-genome scan for association to disease, making a whole-genome scan practical with 1,000-3,000 markers. (2) Control samples are not strictly required for the study, since the proportion of ancestry at each locus is being compared to a genome-wide average to look for a deviation—the control is the rest of peoples' genome. In general, we feel that as a matter of study design, it is far more important to have as large a case sample size as possible, with the size of control samples of secondary importance. For a detailed discussion on this topic please refer to the paper.

Theoretical calculations demonstrate that in many cases, with a high-density map of markers admixture mapping study has statistical power similar to that of a whole-genome haplotype or direct association study. And fewer samples are required than

for a linkage scan to achieve the same statistical power. The key, however is that admixture mapping works best for alleles which have high frequency differentiation across populations.

The power calculations in the accompanying paper suggest that with 2000 samples and a high density map it should be theoretically possible to use this approach to detect disease loci where the relative risk due to an allele is as low as 1.5. We find that samples with population A ancestry between 10%-90% provide the most power for admixture mapping, and that the power is affected mildly by which population has a higher incidence.

3. How to run the program

This section describes how to run the program through the command line, and a description of the input parameter file needed to run it.

a) Command-line arguments

To run ANCESTRYMAP type on the command line:

```
>>ancestrymap –pv paramfile or
>>./ ancestrymap –pv paramfile
```

p: is a compulsory option, and in this case we have to specify the parameter file *paramfile*.

v: version number, this tells us which version of the program we are using. This number can be modified by the user in the file ancestrymap.c . To redirect the output to a file one would type on the command line:

```
>>./ancestrymap –pv paramfile > out.dat&
```

b) Description of the parameter file

The format of this file is as follows:

Parname: parvalue

```
>>seed:      200
```

Note: All the parameter names should be in lowercase, and there should be no white space between parname and semicolon. The parameters which are compulsory are the names of the files that contain marker, individual and genotype data; and the risk model. Parameters which are of the type array should have their values space separated. A sample parameter file is included as part of the download, and a detailed description of the parameters is as follows:

Parameter Name	Data type	Description	Possible and Default values
INPUT FILE NAMES			

indivname (MANDATORY)	String	Individual data	
badsnpname	String	List of markers to delete from analysis	
genotypename (MANDATORY)	String	Genotype data for all the samples	
snpname (MANDATORY)	String	Marker data	
ANCESTRYMAP PARAMETERS			
risk (MANDATORY)	Double array	Risks for the various models	Default: 2.0
numiters	Int	Number of follow-on iterations	Positive integer >= 0 Default: 5
numburn	Int	Number of burn-in iterations	positive integer >= 0 Default: 1
reestiter	Int	Controls number of iterations inside ancestrymap for allele freq sampling	positive integer >= 1 Default: 1
details	Boolean	If YES generate additional output	NO, YES Default: NO
tlreest	Int	Always set to YES, don't need it	0,1
noxdata	Boolean	If you have no X chromosome data or want to ignore it	NO, YES Default: NO
fakespacing	Double	The spacing between fake markers in Morgans	positive > 0 Default: 0.01 (in Morgans)
seed	Int	Random number needed for the run	Positive integer
checkit	Boolean	If YES runs lots of checks (mostly done initially)	NO, YES Default: NO
thxpars	Double array of size 3	Sets the initial parameters for the prior distribution for θ_X	Default: 40.0 1.0 10.0
thpars	Double array of size 2	Sets the initial parameters for the prior distribution for θ.	Default: 1.0 5.0

lampars	Double array of size 2	Sets the initial parameters for the prior distribution for λ.	Default: 1.0 0.1
lamxpars	Double array of size 2	Sets the initial parameters for the prior distribution for λ_X	Default: 1.0 0.1
dotoysim	Boolean	If YES run simulations	NO, YES Default: NO
markersim	Int	This is the marker number of the disease allele, -1 means none	-1 or positive integer Default: -1
simnumindivs	Int	Generate toy data with *simnumindivs* number, half will be cases, and half controls. Half are female and half are male	Positive integer Default: -1
risksim	Double	In simulation mode risk used to generate data	Default: 1.0
tauscal	Double array	Initial values of t(African) & t(European)	Default: 100 100 (Note this is a lower value than we expect, however we prefer to bias the initial value to be low)
wrisk	Double array	Allows the model to have weights, which are normalized to sum to 1	Default: 1.0
lrisk	Double	In *checkit* mode: leave one marker out in turn and this is the risk that we use (in *checkit* mode: only one model risk is used).	Default: -1.0
controlrisk	Double array	Control risks for the various models	Default: 1.0
risk2	Double array	Risk for ethnic homozygotes for various models, *controlrisk* and *risk2*	Default : -1.0

		are optional, however they should be same number as *risk* if they are specified	
taulsdev	Double	Prior standard deviation for african & European t values	Default: 0.5
taulmean	Double	Prior mean for log10(t) for both African and European	Default: 2.0
allmale	Boolean	Used in simulation mode. If YES it specifies that all the simulated individuals should be men. Need to specify the parameter simnumindivs to make this parameter effective	0,1 Default: NO
allcases	Boolean	If YES all the samples are cases	NO, YES Default: NO
usecontrols	Boolean	If NO controls are ignored	NO, YES Default: YES
pubfmodern	Boolean	Publish ancestral allele frequency estimates, if YES allows publication of modern allele frequencies	NO, YES Default: NO
OUTPUT FILE NAMES (Note that the directory in which the output files are to be generated should exist, else the program will fail)			
trashdir	String	Used only in *checkit* mode: directory to store HMM output	
thetafilename	String	Ancestry information for all individuals	
output	String	Parameter values at every iteration	
pubxname	String	Debug file for a particular marker	
ethnicfilename	String	Average ethnicity	

		(/g) for each marker, averaged over all individuals and iterations
snpoutfilename	String	Detailed marker information
indoutfilename	String	Detailed individual information
freqfilename	String	Allele frequency information for all markers
lambdafilename	String	λ information for all individuals
genotoyoutfilename	String	Genotype data generated in simulation mode
indtoyoutfilename	String	Individual data generated in simulation mode

The software makes it possible to test for several disease models simultaneously. If one is studying a disease for which there is an epidemiological reason to believe that there is higher genetic risk in population A, one might want to test several models for increased risk due to population A ancestry and, simultaneously test one model where population B ancestry confers greater risk. This is implemented by inputting the parameter ***risk*** as an array with values both greater and less than 1, for example:
>>risk: 0.8 1.2 1.3 1.4 1.5 1.6

4. Format of the input data files

In this section we will discuss the format of the input files that are needed for the executable to run. The file names are specified in the parameter file used by the program. The data in all the input files should be white space separated or tab separated.

- **Marker file** (*snpname*):
  This contains information about the markers being used for the analysis. The format and an example of the file is as follows:

SNP_ID	Chr	Gen_ Pos	Phys_ Pos	PopA_ vart_cnt	PopA_ ref_cnt	PopB_ vart_cnt	PopB_ ref_cnt
rs897634	1	0.031621	2618675	21	189	242	84
rs905135	1	0.035690	2982467	35	71	281	19
CV1944294	1	0.067986	4380773	42	64	277	21

Here Chr_Num is the chromosome number, Gen_Pos and Phys_Pos are genetic and physical positions. PopA_vart_cnt and PopA_ref_cnt are the variant and reference allele counts in the parental samples of population A, and the last two columns are these counts for the parental samples of the population type B.

The genetic position can be in Morgans or centiMorgans, and valid values for chromosome_num range from 1 to 23, or 1 to 22 and X. The markers can be arranged in any order in this file, and don't have to be sorted by chromosome number or any other field. Currently the algorithm does not support the Y chromosome, mitochondrial DNA, or the pseudoautosomal region of the X chromosome.

One could alternatively use a file which has only the first four columns, that is, with no parental counts. This will probably lead to reasonable results, however with lower statistical power, and the user should be cautious about the results in this case. If the user has a marker file which has just the first four columns and a genotype file for the parental populations, one can generate the file in the above format using the program cntmono, which is described in detail in Section 11. Before using the output file created by *cntmono* as the input marker file for *ancestrymap,* remove from it the blank lines, lines with comments and, the header line. This file should only contain details about the markers, else *ancestrymap* will give a fatal error.

- **Badsnps file** (*badsnpname*):
  This is a list of markers that one would like to exclude from the analysis. These could be markers that fail any of the tests described in Section 5 that are performed during the initial phases of running ANCESTRYMAP, by setting the *checkit* field to YES in the input parameter file. In addition, one should also exclude one of the pairs of markers which are in strong linkage disequilibrium with each other.

SNP_ID
rs578459
CV2800274
rs73494

- **Individual file** (*indivname*):
  This has information about the individuals that we are going to use for analysis.

Indiv_ID	Gender	Status
I1	M	Control
I2	M	Case
I3	M	Ignore

The gender field can be M (male), F (female) or U for samples with unknown gender. The status field can be Case, Control or Ignore, where the samples that have status set as Ignore are excluded from the analysis. One can use this field effectively without having to create a new individual file each time we want to analyze the same

sample set for a different hypothesis. For example if we have data from case and control samples for multiple diseases (ex. Multiple Sclerosis and Prostate Cancer), and say we want to analyze output from the ANCESTRYMAP only for MS. Then we might want to use the controls for both the diseases as controls, MS cases as cases, and set the Prostate Cancer cases as Ignore. Also, if during the course of analyzing a data set we realize that there is a problem with a particular sample (ex. contaminated DNA) we can set the Status field to Ignore and that would remove this sample from our analysis.

- **Genotype file** (*genotypename*):
  This has the genotypes for all the individuals and markers that are listed in the above two files.

SNP_ID	Indiv_Id	Vart_allele_cnt
rs1865056	I1	0
rs1865056	I2	1
rs1865056	I3	0

Note that there is a fatal error if one has markers and individuals mentioned in the genotype file, which have not been specified in the marker and individual files respectively. The possible values for the variant allele count are 0, 1, or 2. The variant allele count for men on the X-chromosome can be given only as 0 or 1, with 2 being an invalid value in this case. Missing data can be specified by -1, or not mentioned at all. An individual with a large amount of missing data will cause ANCESTRYMAP to behave badly, and it might be a good idea to ignore these individuals in the analysis, by setting their Status field to Ignore.

The genotype file can be given as a zipped .gz file as well, which the program will unzip and use.

5. Built-in data checks

Next we shall focus on the built-in data checking programs. Like most other methods for whole genome scans, admixture analysis is very sensitive to data problems and the software incorporates a number of tools to check for the more common kinds of errors. The user is strongly advised to run these tests because our experience suggests that most data sets even when carefully curated contain some problems which can lead to spurious associations to disease. In order to run these checks on the data, you have to run the ANCESTRYMAP program with the *checkit* parameter set to YES in the parameter file. The description of these tests and their output is as follows:

- ***Hetxcheck***:
  Check to see if there are any heterozygous counts on the X chromosome for the male samples**.** The program will disregard heterozygous genotype value for male samples on the X chromosome, if there are any. The output from this check is as follows for all the markers:
  SNP_ID NUM_HET NUM_HOMOZY

>> hetxcheck rs211644 0 310

Here NUM_HET and NUM_HOMOZY are the number of heterozygous and homozygous counts respectively on the X chromosome for the male samples.

- ***checkgeno***

  Checks to see if there are any genotype values > 2, prints out a warning and ignores that genotype for the rest of the analysis.

  >>bad genotype: rs897634 1 4

  This test also outputs the total number of good and bad genotypes. Ex:

  >>Num good genotypes: 4711298 Num bad genotypes: 0

- ***physcheck***

  This is a check to find markers which are flipped with respect to their genetic and physical positions. The output is only for the set of two markers, where there has been a mix up of physical and genetic positions. This check gives a warning only, since we use genetic distance in the analysis, and not the physical distance.

  SNP1_Id SNP2_Id SNP1_Gen_Pos SNP2_Gen_pos SNP1_Phys_Pos SNP2_Phys_Pos

  >> physcheck rs11231098 rs435582 0.628 0.649 61996439 41813770

- ***Hardy-Weinberg test***:

  Performs the hardy-weinberg equilibrium test for each marker and prints out:

  SNP_Id Chr_Num SNP_Index HW_score.

  >> hwcheck rs897634 1 0 -1.526

  A positive *HW_score* is indicative of too many heterozygous counts, and a negative score is indicative of too many homozygous counts. For markers that are highly differentiated in frequency, a deficit of heterozygotes is often observed in a population such as African Americans (this is called the Wahlund effect in population genetics). Thus a hwcheck result showing an excess of heterozygotes should be a greater cause for worry than one showing a deficit. One should look for outliers in this test.

- ***checkdup***

  This checks individuals to see if there are any duplicate individuals based on the amount of match between their genotypes. If there is more than a 75% match in the genotypes for two samples, this test prints out:

  >>##Num of genotypes matched: Num of genotypes mismatched

  >>##If the status of the two individuals does not match Status1: Status2

  >>dup? Indiv_1 Indiv_2

  >>match: 1400 mismatch: 2

  >>status_1: Case status_2: Control

  The above example indicates that these two individuals are probably the same since their genotypes match exactly. Note that the test also prints out the status of the two individuals if they do not match.

However, the next example shows that even though there might be say 80% match between two individuals, they might not be duplicates since the number of genotypes compared is not very large. The user has to look at the results carefully and decide which pair of individuals are duplicates and which are not. The user should set their own cutoff of what defines a duplicate pair of individuals.

>>dup? Indiv_2 Indiv_35
>> match: 100 mismatch: 20

New in Version 2.0: *fastdup* parameter

We allow for a very fast, but far from comprehensive check for duplicate samples. The basic algorithm chooses 15 markers and looks at genotypes on these 15 markers. Pairs of individuals with an exact match on the 15 markers are checked for near duplicates with a slow algorithm that counts matches and mismatches for every marker. We iterate this check *fastydupnum* times (default for *fastdupnum*: 10).This check is very fast and has a reasonable chance of finding duplicates, but can be defeated by missing genotypes, or genotype errors. To run this check set

fastdup = YES
dupmode:= YES

for a careful check for duplicates with running time proportional to the square of the number of samples. The output for this check prints out the duplicate pair IDs, # matches, # mismatches, # valid genotypes for each individual in the pair, and then automatically "ignores" one of the samples. The user will have to set one of the samples to 'Ignore' in the sample file on their own.

>>dup? Indiv_1 Indiv_2
>>match: 665 mismatch: 0 1450 1495
>>dup. Indiv_1 ignored

- ***mapcheck***:

  Compares ancestry estimates obtained for each marker by itself to that predicted by adjacent markers (leaving out the marker of interest). A discrepancy indicates a misspecification of a marker's genomic position. A negative difference is not worrisome however a positive difference should be investigated more carefully, especially if it is higher than 3 or 4. Note that for this test it is more important to look for outliers than at absolute values alone.

  SNP_ID SNP_Index Ancestry Difference
  >> mapcheck rs897634 0 -23.679

  Here SNP_Index is the marker's index number internal to the program.

- ***freqcheck***

  Freqcheck compares the estimated frequencies of an allele from the MCMC (Markov Chain Monte Carlo) with a max likelihood fit. S() is a likelihood ratio statistics, approximately chi-square with 1 d.o.f. if the frequencies look fine. This is really a check that the parental frequencies are plausible. S scores above 10 are highly dubious, above 20 indicate a problem .A common reason

for this error is due to an interchange of alleles (flipped marker). Same as in the case for mapcheck it is more important to look for outliers.

SNP_ID Chr_num S(All) S(Controls) F(A) F(E) G(A) G(E)
>> freqcheck rs897634 1 1.133 2.563 0.086 0.765 0.062 0.847

Here S(All) is on all the data, and S(Controls) is on controls only, as a very strong disease effect in cases can distort the true frequency. F(A), F(E) are estimated frequencies for the African and European parental samples using the MCMC, G(A), G(E) are the corresponding maximum likelihood fits.

- ***leave1out***:

  Removes the marker contributing the most to any association and assesses whether the signal of association persists. If it remains even after leaving out the best marker, it is less likely to be an artifact due to a single marker. This is a computationally expensive check to run, and needs a large amount of disk space and might crash if that is not available.

  >> scores for each fake
  >> chrom SNP_ID base min max
  >> 1 1 fake-1:0 -6.912 -7.063 -5.286
  >> 3 1 fake-1:1 -6.965 -7.169 -5.389
  >> 4 1 fake-1:2 -6.752 -7.050 -5.425

  Here base is the score that we get without using the leave1out algorithm, min and max are the minimum and maximum scores obtained after leaving one marker in turn. The max score is not relevant, however it is a cause of worry if the min score is very much lower than the base score.

  This test gives the following output as well for all the chromosomes:

  >>chrom base min max
  >>best score (chrom) 1 16.761 14.245 16.875
  >>best score (chrom) 2 -2.101 -2.829 -1.140
  >>best score (chrom) 3 -3.416 -3.645 -1.207
  >>best score (chrom) 4 -3.910 -4.308 -2.679
  >>best score (chrom) 5 -1.444 -1.687 -0.438

  >>best score (chrom) 22 -2.006 -6.231 -1.065
  >>best score (chrom) 23 2.736 -4.751 17.266
  global score (leave1): 13.208

- ***checkindiv***

  New in Version 2.0

  This implements a crude check on whether an individual should be included in the scan, using the idea of estimating global ancestry (proportion of European ancestry for African-Americans). Given the variant allele frequency conditional on ancestry for marker k we can compute the probability distribution of 0, 1, 2 variant alleles and hence a log-likelihood score $L(k)$. We also can compute the mean and variance of $L(k)$. Accumulating the statistic $L(k)$ over all markers k we get a statistic whose mean and variance is known. Therefore we can compute a

Z-score, large negative scores (say < -6) should be discarded. In practice we also find large positive scores. These individuals usually have parents with very divergent ancestries, whose children therefore have, marker by marker ancestry close to the mean. We recommend that such individuals are also not used in the scan, though this is a minor issue as they will not contribute much to the admixture score. Here is some output from samples that we would not use in a scan. Note that the top 3 individuals have ancestry proportions near 50%.

>>###	***ID***	***P(E)***	***---***	***Z-score***			
>>checkindiv	***Hi1***	***0.504***	***156.756***	***10.464***	***1328***	***888.585***	***0.669***
>>checkindiv	***Hi2***	***0.514***	***143.433***	***10.451***	***1149***	***730.872***	***0.636***
>>checkindiv	***Hi3***	***0.506***	***146.305***	***10.286***	***1225***	***824.703***	***0.673***
>>checkindiv	***Lo1***	***0.360***	***-103.759***	***-6.970***	***1128***	***1604.437***	***1.422***
>>checkindiv	***Lo2***	***0.364***	***-131.407***	***-9.242***	***1026***	***1639.528***	***1.598***

6. **Interpreting and monitoring the output**

a) Overview

In this section we will discuss the output generated by the ANCESTRYMAP program, and how to analyze it.

At the end of a run the most important output generated by the ANCESTRYMAP program is the log-likelihood of the locus genome statistic (LOD) averaged over all the markers in the genome, referred to as the genome log-factor. As mentioned in the paper LOD > 2 is a significant score, and is a signal of positive disease association. At the end of the run one should check this value and also the τ values. If the τ values are small (<100) it is an indicator of problems: in particular that the ancestral population frequencies are not matching up well with the admixed population frequencies.

In our experience of running this program one can observe a high score of association, due to many reasons that have nothing to do with a real association to disease. Thus, if a significant LOD value is observed the user should be extremely careful about interpreting the result and first perform the following checks:

- Is the rise (or fall) in population A ancestry that is suggesting the presence of a disease gene seen in both cases and controls? If it is, this suggests an artifact affecting both cases and controls, rather than a real biological association to disease.
- Does one observe a significant case control statistic score at the locus?
- Have any markers within 10cM of the disease locus failed any of the data checks described in Section 5 ? If so, remove them from the analysis (add them to the *badsnpname* file) and rerun the program.
- Remove the individual marker showing the strongest association to disease from the analysis by adding it to the *badsnpname* file and then rerun the program. The score should not be dependent on a single marker, but rather be supported by multiple independent markers.
- In our experience, markers that are in linkage disequilibrium in the ancestral

populations (E.G. West Africans and European Americans), but are treated as independent for the admixture mapping study, can produce a false-positive association to disease. This is especially important because it is tempting to increase marker density, thereby the chance of markers being in linkage disequilibrium with each other in the ancestral populations, in precisely the regions that show most association to disease in preliminary scans. We therefore recommend testing for linkage disequilibrium in the ancestral population among the markers within 10cM of the disease locus, and discarding markers until no pairs of markers in the region are in linkage disequilibrium with each other ($P<0.05$). At markers showing association to disease, thin the markers so that none is within 200 kb of any of the others. Finally, rerun the analysis to assess if the association remains.

- One should also perform a few more runs, increasing the number of burn in and follow on iterations by a factor of 10 to assess if the score is consistent.

b) Output Details

Next we will discuss the standard output in detail, which can be redirected to a file for easier viewing. There are a number of parameters that control the output. The most basic output generated is as follows (when no output parameters have been set) to the standard out, which can be redirected to a file for easier viewing.

- Input parameter file name
- Values of all the parameters specified in this file
- Total genomic distance
- Count of individuals, cases, controls and ignores used in the analysis; and also the number of real and fake markers
- Score generated by the expectation maximization algorithm for each iteration. One should observe the score increasing with the number of iterations. Ex.:

  >>emsimple iter: 3 53989.104

- Results of the Markov Chain Monte Carlo iterations, which include stimation of θ and λ . Note that the iteration number goes from 1 – *numburn* to 0 for the burn-in iterations and from 1 to *numiters* for the follow-on iterations. Also the score is zero for the burn-in iterations, since we calculate it only for the follow-on iterations.

  estglob theta: iteration_num, thp1, thp2, thxp1, thxp2, thxp0
  estglob lambda: iteration_num, lp1,lp2,lxp1,lxp2, average λ
  thp1, *thp2* are the parameters for the prior distribution for θ, and *thxp0*, *thxp1* and *thxp2* are the same for θ_x Similarly, *lp1*, *lp2* are the parameters

for the prior distribution for λ, and *lxp1*, *lxp2* are the same for λ_X. Ex.:

```
>> estglob theta    30    1.869    7.342      1.169    7.300    47.216
>> estglob lambda   30   21.173    3.528      7.237    1.525     5.998
```

- Posterior estimates of the mean and standard deviations of θ, θ_x, λ, λ_x, τ*(Afr)* and τ*(Eur)*. The user should look at the values of τ (African) and τ (European) carefully, since they are an indicator of how well the ancestral models fit the data. It is worrisome if we see values to be less than 100.

- Genome-wide scores for all the models, ex.:

```
>> risk1     risk2     crisk    score
>> model: 1.500  2.250    1.000   9.028
```

  Here, risk1 and risk2 are the increase in risk corresponding to have one or two population A ancestry alleles (as opposed to having none), and crisk is the corresponding control risk.

- Lag and correlations For a number of sample statistics we compute correlation coefficient at small "lags". If the statistic at iteration i is S(i) we compute for 1 <= lag <= 10 (default) the correlation between S(i) and S(i+lag). Large values indicate that the MCMC is not mixing very well. We publish this for:
  - llike: a statistic of no intrinsic interest but mixes poorly.
  - log10fac: Log_10 Bayes factor (genome wide)
  - factor: Bayes factor = 10^log10fac
  - log tauscal: log (t(0)) the t value for population 0.

  In our experience ii), iii) are the most important statistics which mix well, iv) mixes less well and i) mixes quite poorly. Ex.:

```
>>  llike mean: 16394.546 s.dev:  2397.753
>> lag:  1 corr:    0.781
>> lag:  2 corr:    0.741
>> lag:  3 corr:    0.656
```

- Scores for each chromosome:
  Chr_Num LGS_MAX CCS_MAX CCS_MIN LGS_LOCAL. *LGS_MAX* is the maximum locus genome statistic score obtained on the chromosome, *CCS_MAX* and *CCS_MIN* are the maximum and minimum case-control statistic scores, and *LGS_LOCAL* is the log likelihood of the locus genome statistic score calculated by averaging over all the markers on that chromosome. All of these scores are averaged over all the individuals and all the iterations. Here is an

example:

```
>>1   1.40    2.22  -2.04   -0.11
>>2  -0.53    2.27  -2.56   -1.68
>>3  -1.39    1.17  -2.31   -2.55
>>4  11.89    5.66  -1.40   10.28
>>5  -0.78    0.74  -1.93   -1.78
>>6   0.11    2.49  -2.34   -1.36
>>7   0.41    2.18  -2.79   -0.77
>>8  -1.64    1.73  -0.87   -2.81
>>9  -2.51    1.00  -2.01   -3.46
>>10 -1.58    0.31  -2.11   -3.01
>>11  0.22    1.38  -0.75   -1.12
>>12 -0.03    2.71  -0.18   -1.02
>>13  0.09    0.85  -1.16   -1.34
>>14 -1.92    0.04  -3.26   -2.88
>>15 -1.28    2.07  -0.66   -1.93
>>16  1.71    2.73  -1.50    0.45
>>17 -1.70    1.08  -1.41   -2.72
>>18  0.81    3.30  -0.77   -0.21
>>19 -1.35    1.17  -1.99   -2.28
>>20 -0.33    2.51  -1.17   -1.28
>>21 -2.51    0.81  -0.66   -3.43
>>22 -2.24    0.64  -0.62   -2.97
>>23  0.00    1.39  -2.27   -1.36
```

As one can clearly see from the above example, the *LGS_MAX* and *CCS_MAX* scores are the highest for chromosome number 4.

- Bestscores: The maximum genome-wide score for the locus-genome statistic, and the maximum and minimum genome-wide scores for the case-control statistic.

```
>>bestscores:    11.894    5.658   -3.263
```

- Genome-log-factor: log-likelihood of the locus genome statistic averaged over all the markers in the genome.

```
>> genome log-factor:   9.028
```

The genome log factor is the most important output of ANCESTRYMAP and should be the first number that the user looks at.

Next we will discuss the output when the parameter *details* = YES in the parameter file. In this situation the following additional output is written to the screen:

- Details about all the chromosomes C**hr_Num First_snp Last_snp Gen_dist**. Here *Chr_Num* is the chromosome number, *First_snp* and *Last_snp* are the indices of the first and last markers on the chromosome and *Gen_dist*

is the genetic distance. Ex:

```
>> 2   401    754      2.552
```

- For each markov chain monte carlo iteration print out, in the following format: The format of the output which begins with bigiter is as follows:
  bigiter iter ylike LOD sc. tau(A) tau(E) thetaave lambdaave xtave(M) xlave(M) xtave(F) xlave(F)
  ylike is the slowly mixing statistic of little intrinsic interest described in the above mentioned supplementary note. xtave(M) and xtave(F) are the average θ on X chromosme for males and females respectively, xlave(M) and xlave(F) are the average λ on X chromosome for males and females respectively. Ex:

```
>> bigiter:   30      14677.057    8.567    84.266    99.386     0.199
5.996     0.187     4.624     0.196     4.730
```

- θ/M and λ values for all individuals:
  Indiv_Index Indiv_ID Gender tmean tsdev txmean txsdev lmean lsdev lxmean lxsdev.
  Here *Indiv_Index* is the individual's internal index number, *tmean* and *txmean* are the average θ and θ_x, *tsdev* and *txsdev* are the standard deviation for θ and θ_x. *lmean* and *lxmean* are the average λ and λ_x, *lsdev* and *lxsdev* are the standard deviation for λ and λ_x.

```
>> 24         toyindiv:24 M     0.458     0.030     0.410     0.068
8.118     0.989     4.921     1.320
```

- Allele frequency estimates with standard error:
  SNP_Index SNP_Id Chr_Num amean asdev bmean bsdev.
  Here *SNP_Index* is the internal snp index, *amean* and *bmean* are the average reference allele frequency in population A and B; and *asdev* and *csdev* are the corresponding standard deviation.

```
>> 12   rs3007429          1     0.597     0.014       0.064     0.021
```

- Scores for each marker:
  SNP_Index SNP_Id Chr_Num Phys_Pos Gen_Pos LGS CCS.
  Phys_Pos is the physical position, *Gen_Pos* is the genetic position, *LGS* is the locus genome statistic score and *CCS* is the case-control statistic score. Both the scores are averaged over all the individuals and iterations.

```
>> 11   rs1534997       1   5953261       0.111    -6.869     -2.010
```

For a more detailed discussion on output details in various scenarios please refer to the online documentation Output Details section.

c) Output Files

Next we will discuss the format of the output files, which can be specified in the parameter file. For an explanation of the format of these files click here.

- ***indoutfilename*** specifies the following information for all the samples analyzed:
  - Indiv_Id
  - Gender
  - Status
  - Num_valid_genotypes

- ***snpoutfilename*** specifies the following information for all the markers analyzed:
  - Snp_Id
  - Chromosome_num
  - Genetic_pos
  - Physical_pos
  - Pop_A_variant_allele_count
  - Pop_A_ref_allele_count
  - Pop_B_variant_allele_count
  - Pop_B_ref_allele_count
  - Case_genotype_count
  - Control_genotype_count

- ***thetafilename*** specifies the following information for all the analyzed samples:
  - Indiv_index
  - Indiv_id
  - θ_true: "true" value of θ or M, printed only in the simulation mode
  - θ_ mean: population A ancestry for the autosomes averaged over all the iterations for a particular individual
  - θ _sdev: standard deviation of θ_ mean
  - θ_X_true: "true" values of θ_X or M_X, printed only in the simulation mode
  - θ_X_mean: population A ancestry for the X chromosome averaged over all the iterations for a particular individual
  - θ_X_sdev: standard deviation of θ_X_mean
  - Status

- ***lambdafilename*** specifies the following information for all the analyzed samples:
  - Indiv_index
  - Indiv_Id
  - λ_true: "true" value of λ, printed only in the simulation mode
  - λ_ mean: λ for the autosomes averaged over all the iterations for a particular individual
  - λ_sdev: standard deviation associated with λ_ mean
  - λ_X_true: "true" value of λ_X , printed only in the simulation mode
  - λ_X_mean: λ for the X chromosome averaged over all the iterations for a particular individual
  - λ_X_sdev: standard deviation associated with λ_X_mean

- ***freqfilename*** specifies the following information for all the markers analyzed:
  - SNP_Index*:* index internal to the program for the snp
  - SNP_ID
  - chromosome_num
  - atrue: "true" reference allele frequency in population A, valid only in simulation mode
  - anaive: naïve frequency of the reference allele in population A using the ancestral genotype data
  - amean: calculated frequency of the reference allele in population A averaged over all the iterations
  - asdev: standard deviation associated with amean
  - btrue: "true" reference allele frequency in population B, valid only in simulation mode
  - bnaive: naïve frequency of the reference allele in population B using the ancestral genotype data
  - bmean: calculated frequency of the reference allele in population B averaged over all the iterations
  - bsdev: standard deviation associated with bmean

- ***ethnicfilename*** specifies the following information for all the markers:
  - SNP_Index
  - chromosome_num
  - SNP_ID
  - Avg_ethnicity***:*** Average θ or M over all iterations, and over all individuals at a particular marker.

- ***pubxfile***: Contains ancestry estimates for either a single marker or individual depending on the usage. In either case it outputs the probability of having 0, 1 or 2 PopB chromosomes in the columns G[0],G[1] and G[2].

- **localoutfilename**: contains the scores for all the markers:
  - SNP_Index
  - Chromosome_Num
  - Physical_Pos
  - Genetic_Pos
  - Log Genome Score
  - Case Control Score
  - G(Case) : Average ancestry for all cases at that marker
  - G(control) : Average ancestry for all controls at that marker
  - rpower: Information content

- **output**: This is the output file which has the following information for all the Markov chain monte carlo iterations:
  - Iteration_Num
  - θ_mean
  - θ_x_mean
  - θ_corr

- λ_mean
- λ_x_mean
- λ_corr
- τ(popA)
- τ(popB)
- log score
- log score averaged over iterations

Note that if this file name is not specified in the parameter file, we write the above to the standard output.

The following two files are written to when we run the program in the simulation mode:

- **Genotoyoutfilename**: specifies genotype data for all the markers and simulated individuals in simulation mode:
  - SNP_ID
  - Indiv_ID
  - Vart_allele_count

- **Indtoyoutfilename**: specifies the following information for the simulated individuals in simulation mode:
  - Indiv_ID
  - Gender
  - Population

If the program is run with *checkit* = YES, then the results of the data check programs mentioned in Section 5 are directed to the standard output.

As detailed in the paper we feel that 100 burn-in iterations and 200 follow on iterations should be sufficient for most analysis. These are the number of suggested iterations for most exploratory runs, and user can increase these numbers in order to confirm the results. One can plot the genome-wide-score as a function of iteration number, to see how well the score converges.

7. Enhancements in the Version 2.0

The main enhancement in this version is the ability to do a fine-mapping run to follow up a peak in a coarse scan. In addition to doing the fine-mapping run one can also run simulations in this mode. A number of small enhancements have been added as well, such as fast check for duplicates, support for PED files and some extra output. A number of small bugs have been fixed as well.

In detail the main enhancements are as follows:

1) A new program ***baseprog*** allows the user to run the parameter file to make sure the input files are valid, but does not do the MCMC calculation. See the example

parameter file parbaseprog which is part of the download. To run the program type on the command line:

>> baseprog –p parbaseprog or

>>./baseprog –p parbaseprog

p: is a compulsory option, and in this case we have to specify the parameter file *parbaseprog*.

To redirect the output in a file one would type on the command line:

>>./baseprog –pv parbaseprog > outbaseprog.dat&

2) Fine-mapping run capability including simulations. A detailed write up is included in the next section.
3) Fast duplicate check: This is a quick check looking for duplicates in the dataset, which is particularly useful in cases where we have a large number of samples, some of which maybe duplicates. One can do this check by setting the parameter *fastdup* = YES in the parameter file.
4) Checkindiv check: This is a check for discarding individuals from the run, automatic in *checkit* mode.
5) One can use the physical position information to calculate or reset the genetic position, useful for cases where user doesn't have the genetic positions. This is implemented using the parameter *usephyspos*.
6) There is a new pack mode which supported using the parameter *packmode*. By default the program sets this to YES, if number of genotypes is greater than a certain number. The user should set this parameter to YES if memory requirement seem large.
7) User can specify the high and low boundary values for log scores, through the parameters *hiclip* and *loclip*.
8) PED file support, look at the documentation included.
9) One can print ancestry estimates for a particular SNP or Sample using the parameters: pubxindname, pubx, pubxa and markername. For ex. If user wants to dump gammas for a particular individual with internal individual index 2904 into a file called gammaoutfile, one would specify the following parameters:

   pubxname: gammaoutfile

   pubx: 2904 -1

   To dump gammas for a particular marker:

   pubxname: gammaoutfile

   markername: fake-1:1672

   Ex. gamma output files for individual and marker.
10) We now print out information content for a SNP, this is the rpower column in the output file.
11) The user can print out the scores for all the markers in a file, to use this functionality the user will need to specify the parameter *localoutfilename*.
12) New parameters with this release are given in the accompanying table.
13) MAC Release: Experimental version not as well tested is available for download.

8. Fine-Mapping Runs

a) Overview

After carrying out an admixture scan, (say with the methods we have implemented in ANCESTRYMAP and described in [2]), it is essential to follow up in areas of the genome that have promising association scores. As the admixture peaks will be wide,

containing perhaps 100 genes, the real biological pay-off will come from fine-mapping that is in identifying the actual variant in the region that causes disease risk. We sketch out a strategy for doing this in the very same samples initially used for admixture disease gene localization.

The strategy for fine-mapping follow-up is to genotype a large number of SNPs (more than 1,000) across the peak of admixture association, at a resolution of one every few thousand base pairs. In an analysis using African-American data, the goal is to identify a SNP that is in strong LD with the disease-causing variant in the African and/or European ancestral populations. This LD will be inherited in African Americans, and will permit the accurate localization of the disease gene to within a few tens of kilobases.

To be more specific, we note that in our published paper on admixture mapping methods [2], we introduced a log factor score for a given risk model, and locus. We review this scoring method for African Americans, which have experienced a history of admixture between European and African populations, though the methods are of course quite general. Assume for the moment that all model parameters, such as the average amount of European ancestry of each individual, and the allele frequencies of all markers in the parental populations are known exactly. Next, we assume a risk model, and locus, so that if an individual has *a* chromosomes of European ancestry at that locus, then the risk factor is $\psi(a) = P(D|a)/P(D|0)$, where P (D) denotes the probability of disease. (Our notation allows European ancestry to be protective as well as more risky, so that $\psi(a) < 1$ if $a > 0$.) In [2] we showed that the log-factor L(i) for individual i, for our causal hypothesis against the null hypothesis that $\psi(a) = 1$ for each *a* is:

$$\mathcal{L}(i) = \log \sum_{a=0}^{2} \gamma(a,i)\psi(a) - \log \sum_{a=0}^{2} \theta_i(a)\psi(a) \qquad (1)$$

Here $\gamma(a, i)$ is the probability that individual i has *a* European chromosomes at the locus, given all our observations, and $\theta i\ (a)$ is the average ancestry for the individual. The overall log-factor for all the samples is then just the sum of the log-factors over all individuals.

We now show that we can extend our methods to form a score suitable for fine-scanning. In the above theory, if ancestry at a locus is known, then the alleles at the locus are irrelevant to disease risk. But this will not be the case if the marker is in LD with a risk allele, except in the extreme case that the allele is a perfect surrogate for ancestry. (In African Americans only one such example is known: the 'Duffy' null allele [1].)

Generalizing the above theory, we introduce a risk function $\psi(a,b)$ for an individual with *a* European chromosomes and *b* variant alleles. Then we can generalize equation (1) in this case to:

$$\mathcal{L}(i) = \log \sum_{a=0}^{2} \gamma(a,i)\psi(a,b) - \log \sum_{a=0}^{2} \theta_i(a) \sum_{c=0}^{2} B(c|a)\psi(a,c) \qquad (2)$$

where b is the number of variant alleles actually carried and $B(c|a)$ is the conditional probability of c variant alleles given a European chromosomes. It is easy to check that if the risk function $\psi(a,c)$ is independent of c, then (2) reduces to (1) as it should. In practice $\psi(a,c)$ is unknown to us. We next discuss the values of which values of ψ to try. As an ansatz, set

$$\psi(a,b) = e^{a\lambda} e^{b\mu}$$

where we will now choose λ,μ. Fix μ for the moment and assume that the ancestry risk for one copy of a population 2 allele is r, and that r is known, or at least tightly estimated. In practice we are fine-mapping after an coarse scan 'hit' from ANCESTRYMAP so this is not unreasonable. Now it is easy to check that

$$r = \frac{\sum_{b=0}^{2} P(b|a=1)e^{b\mu}\lambda}{\sum_{b=0}^{2} P(b|a=0)e^{b\mu}}$$

so that

$$e^{\lambda} = e^{\lambda(\mu)} = \frac{r\sum_{b=0}^{2} P(b|a=0)e^{b\mu}}{\sum_{b=0}^{2} P(b|a=1)e^{b\mu}}$$

Setting $\mu = 0$, yields the model in which only the ancestry is relevant, and not the genotype. Given this easy theory, we can now readily compute a logfactor $F(\mu)$ for a given hypothesis for the value of μ, against a null hypothesis that $\mu = 0$. We in practice set μ on a mesh spaced uniformly on a log-scale and then factor-average. The strategy is then to see if a substantially larger Bayes factor can be found by allowing μ to be non-zero, indicating an additional effect of the SNP above and beyond the admixture association.

Suppose then that we have N mesh points both for $\mu,\lambda(\mu)$ and if (λ_k, μ_k) are the k-th λ, μ pairs we choose on the mesh, and F (k) is the Bayes factor we obtain for the k-th such pair, a natural score for the fine-mapped locus is $\sum_k \mathcal{F}(k)/N$ This is a likelihood ratio for the hypothesis that one of our mesh points is correct, against the null where $\mu = 0$; that is, there is no contribution of the allele above and beyond the admixture association.

b) Setup of Runs

Once you have successfully run the coarse scan and obtain a peak local log factor > 4 consistently, one can then proceed to "bombard" the region with more SNPs for the same samples in order to find a causal allele. Once you have the genotype data for that you can proceed to use the fine-mapping part of the software, which will give the extra allelic risk on top of ancestry risk. In order to do this properly, one must make sure NOT to include the fine-mapping SNPs in the main coarse scan run. An easy way of doing this is to include the fine-mapping SNPs in the badsnp file. The reason for this is that usually one would have chosen a number of SNPs very close to each other on the same chromosome. These markers are usually in linkage disequilibrium with each other, and as already discussed in Section , SNPs in LD can

lead to false positive scores. After doing the coarse scan, identify the peak region and then start a few runs around the peak region using the fine-mapping parameter file.

It is important that a fine mapping SNP is not in LD with a framework SNP in the parental populations. It is a little complicated to achieve this, we do it by making a series of runs in which we fine map over say a megabase region with no framework SNP in the region or very close by. We provide an annotated Perl script mkfine, and an accompanying template parameter file which the user should be able to modify for his/her requirements. The script mkfine has a number of parameters which the user will need to set depending on their individual scan. It is important to read the script carefully before starting to use it. The script can be looked at in detail in the *bin/* directory, and the accompanying template parameter file parfine.templ is also in this directory.

Once you have set the various parameters in the 2 files, you can just run the script mkfine by typing on the command line:
>> perl mkfine

This should start the various ANCESTRYMAP runs in the fine-mapping mode, once all the runs are over you can take a look at the output files to see if there is anything interesting.

c) Fine-Mapping Output

In this section we will discuss in detail the output generated to standard output, in the case where *details* = NO, *checkit* = NO; with finite number of burn-in and follow-on iterations. This output can be redirected to a file for easier viewing.

- Input parameter file name
- Values of all the parameters specified in this file
- Total genomic distance
- Count of individuals, cases, controls and ignores used in the analysis; and also the number of real and fake markers
- Score generated by the expectation maximization algorithm for each iteration. One should observe the score increasing with the number of iterations.
- Results of the Markov Chain Monte Carlo iterations, which include estimation of θ and λ. Note that the iteration number goes from 1 – *numburn* to 0 for the burn-in iterations and from 1 to *numiters* for the follow-on iterations. Also the score is zero for the burn-in iterations, since we calculate it only for the follow-on iterations. The format of the output is as follows:
  estglob theta iter a1 b1 a2 b2 c2
  estglob lambda iter p1 lambda1 p2 lambda2 lambdave
  These are "global parameters" (affect every individual). See supplementary note

2 of the Patterson et. al. 2004 paper for definitions.
lambdaave is the average λ across individuals.

- Posterior estimates for the mean and standard deviation of θ, θ_X, λ, λ_X and τ(Afr), τ(Eur). The user should look at the value of τ(African) and τ(European) carefully, since they are an indicator of how well the ancestral models fit the data. It is worrisome if we see these value to be less than 100.

- Genome-wide scores for all the models

- Theta and Lambda estimates with standard error for all the samples

- Allele frequency estimates with standard error for all the markers

Here Mu is the genotype risk, and lambda is the allelic risk. For a single copy of a chromosome with local ancestry *a* and *b* variant alleles the risk is taken to be exp(*a* lambda) exp(*b* mu). In the table shown below, given Mu, lambda is chosen so that the ancestry risk if the allele is unknown is that specified by the risk parameter of the (coarse scan) model, for example the risk here is 1.5 (see Overview section). The LogScore column (clipped, so the score will not be below 0) is a LOD score for the fine-mapping model against the model where genotype does not correspond to risk. Note that a positive LogScore is a hint of a causal allele. The reader, as a check on understanding, should note that if mu = 1, then the score must be 0 also as is true in the tableau below (row 15).

lmbayes is a Bayes factor averaging over all fine mapping markers in the run. This really needs adjusting by a prior for whether there is a causal marker in the region.

###	Iteration_Num	Mu	Log_Score	Caltd_Lambda
lmdetails	0	0.333	-8.000	0.811
lmdetails	1	0.359	-8.000	0.839
lmdetails	2	0.386	-8.000	0.868
lmdetails	3	0.415	-8.000	0.900
lmdetails	4	0.447	-8.000	0.934
lmdetails	5	0.481	-8.000	0.970
lmdetails	6	0.517	-8.000	1.009
lmdetails	7	0.557	-8.000	1.050
lmdetails	8	0.599	-7.924	1.094
lmdetails	9	0.644	-6.300	1.141
lmdetails	10	0.693	-4.327	1.192
lmdetails	11	0.746	-2.758	1.246
lmdetails	12	0.803	-1.576	1.304
lmdetails	13	0.864	-0.746	1.365
lmdetails	14	0.929	-0.229	1.430
lmdetails	15	1.000	0.000	1.500
lmdetails	16	1.076	-0.057	1.574
lmdetails	17	1.158	-0.412	1.653
lmdetails	18	1.246	-1.075	1.737
lmdetails	19	1.340	-2.058	1.826

```
lmdetails  20     1.442    -3.369       1.920
lmdetails  21     1.552    -5.009       2.020
lmdetails  22     1.670    -6.941       2.126
lmdetails  23     1.797    -7.978       2.238
lmdetails  24     1.933    -8.000       2.356
lmdetails  25     2.080    -8.000       2.481
lmdetails  26     2.238    -8.000       2.612
lmdetails  27     2.408    -8.000       2.750
lmdetails  28     2.591    -8.000       2.895
lmdetails  29     2.788    -8.000       3.048
lmdetails  30     3.000    -8.000       3.207
###lmscore: Fine-mapping score in addition to the Admix_Score
  ##SNP_ID    LMScore    Chr_Num    Phys_Pos Admix_Score
lmscore:      rs11890727    -0.992  2       114383724    14.382
##lmscbest : Best lmscore in the run
lmscbest:    -0.992
##lmbayes: Bayes factor, averaging over all fine mapping markers in the run
lmbayes:     -0.992
```

- Lag and correlations
  For a number of sample statistics we compute a correlation coefficient at small "lags". If the statistic at iteration i is S(i) we compute for 1 <= lag <= 10 (default) the correlation between S(i) and S(i+lag). Large values indicate that the MCMC is not mixing very well.
  We publish this for:
  - **llike**: a statistic of no intrinsic interest but mixes poorly.
  - **log10fac**: Log_10 Bayes factor (genome wide)
  - **factor**: Bayes factor = 10^log10fac
  - **log tauscal**: log (t(0)) the t value for population 0.

  In our experience ii), iii) are the most important statistics which mix well, iv) mixes less well and i) mixes quite poorly.

- Scores for each chromosome
  As one can clearly see from the below example, the *LGS_MAX* and *CCS_MAX* scores are the highest for chromosome number 3.

- Bestscores: The maximum genome-wide score for the locus-genome statistic, and the maximum and minimum genome-wide scores for the case-control statistic.

- Genome-log-factor: log-likelihood of the locus genome statistic averaged over all the markers in the genome. The genome-log factor is the most important number that is produced by the program and should be the first number that the user looks at.

9.0 Input File Formats and Conversion Program

This file contains documentation of the program ***convertf***, which converts between the 5 different file formats we support. Note that "file format" simultaneously refers to the formats of three distinct files:

- genotype file: contains genotype data for each individual at each SNP
- snp file: contains information about each SNP
- indiv file: contains information about each individual

Below, we document all 5 formats:

- ANCESTRYMAP
- EIGENSTRAT
- PED
- PACKEDPED
- PACKEDANCESTRYMAP

and we explain how to use ***convertf*** to get from one format to another. Note all the example files are in the directory:

ANCESTRYMAP Format:

- genotype file: see example.ancestrymapgeno
- snp file: see example.snp
- indiv file: see example.ind

The genotype file contains 1 line per valid genotype, and has 3 columns:

SNP_ID	Sample_ID	Number of Variant Alleles (0,1 or 2)

Missing genotypes are encoded by the absence of an entry in the genotype file.

The snp file contains 1 line per SNP. There are 4 columns:

SNP_ID	Chromosome_Num	Genetic_Position	Physical_Position

Use 23 for X chromosome. The genetic position can be in Morgans or centiMorgans, and the physical position is in bases.

The indiv file contains 1 line per individual, and has 3 columns:

Sample_ID	Gender	Status

The gender column can be M(male), F(female) or U (unknown). The status column might refer to Case or Control status, or might be a population group label. If this entry is set to "Ignore", then that individual and all genotype data from that individual will be removed from the data set in all convertf output. The name "ANCESTRYMAP format" is used for historical reasons only. This software is completely independent of our 2004 ANCESTRYMAP software.

EIGENSTRAT Format: Used by EIGENSTRAT (both in the 07/23/06 release and in the current release).

- genotype file: see example.eigenstratgeno
- snp file: see example.snp (same as above)
- indiv file: see example.ind (same as above)

The genotype file contains 1 line per SNP. Each line contains 1 character per individual:
0 means zero copies of reference allele.

1 means one copy of reference allele.
2 means two copies of reference allele.
9 means missing data.

The program ***ind2pheno.perl*** in this directory will convert from *example.ind* to the *example.pheno* file needed by the EIGENSTRAT software. To run this script type on the command line:
>> ./ind2pheno.perl example.ind example.pheno

PED Format:

- genotype file: see example.ped *** file name MUST end in .ped ***
- snp file: see example.pedsnp *** file name MUST end in .pedsnp *** convertf also supports .map suffix for this input file name
- indiv file: see example.pedind *** file name MUST end in .pedind ***and Conversion between various formats
  convertf also supports the full .ped file (example.ped) for this input file

Note that, mandatory suffix names enable our software to recognize this file format.
The indiv file contains the first 7 columns of the genotype file (see below).
The genotype file is 1 line per individual. Each line contains 7 columns of information about the individual, plus two genotype columns for each SNP in the order the SNPs are specified in the snp file.
The first 7 columns are:

- 1st column is family ID.
- 2nd column is sample ID.
- 3rd and 4th column are sample IDs of parents.
- 5th column is gender (male is 1, female is 2)
- 6th column is case/control status (1 is control, 2 is case) OR quantitative trait value OR population group label.
- 7th column (this column is optional) is always set to 1.

convertf does not support pedigree information, so 1st, 3rd, 4th columns are ignored in ***convertf*** input and set to arbitrary values in ***convertf*** output. In the two genotype columns for each SNP, missing data is represented by 0.
The snp file contains 1 line per SNP. There are 4 columns:

Chromosome_Num	SNP_ID	Genetic_Position	Physical_Position

Use X for X chromosome. The genetic position is in Morgans, and the physical position is in bases.
The indiv file contains the first 7 columns of the genotype file.
The PED format is used by the PLINK package of Shaun Purcell. See http://pngu.mgh.harvard.edu/~purcell/plink/.

PACKEDPED Format:

- genotype file: see example.bed *** file name MUST end in .bed ***
- snp file: see example.pedsnp *** file name MUST end in .pedsnp ***
- convertf also supports .map suffix for this input file name
- indiv file: see example.pedind *** file name MUST end in .pedind ***

convertf also supports a .ped file (example.ped) for this input file

Note that, mandatory suffix names enable our software to recognize this file format.
example.bed is a packed binary file (2 bits per genotype).
The PACKEDPED format is used by the PLINK package of Shaun Purcell. See http://pngu.mgh.harvard.edu/~purcell/plink/.
For input in PACKEDPED format, snp file MUST be in genomewide order.
For input in PACKEDPED format, genotype file MUST be in SNP-major order (the PLINK default: see PLINK documentation for details.)

PACKEDANCESTRYMAP Format:

- genotype file: see example.packedancestrymapgeno
- snp file: see example.snp (same as above)
- indiv file: see example.ind (same as above)

Note that, example.packedancestrymapgeno is a packed binary file (2 bits per genotype).

DOCUMENTATION OF convertf program:

To run this program type on the command line:
>> /bin/convertf -p parfile

We illustrate how *parfile* works via a toy example: (see example.perl in this directory)
par.ANCESTRYMAP.EIGENSTRAT converts ANCESTRYMAP to EIGENSTRAT format
par.EIGENSTRAT.PED converts EIGENSTRAT to PED format
par.PED.EIGENSTRAT converts PED to EIGENSTRAT format
par.PED.PACKEDPED converts PED to PACKEDPED format
par.PACKEDPED.PACKEDANCESTRYMAP converts PACKEDPED to PACKEDANCESTRYMAP
par.PACKEDANCESTRYMAP.ANCESTRYMAP converts PACKEDANCESTRYMAP to ANCESTRYMAP

Note that the choice of which allele is the reference allele may be arbitrary and thus converting to a new format and back again may change the choice of reference allele.

DESCRIPTION OF EACH PARAMETER in parfile for convertf:

Parameter Name	Data type	Description	Possible and Default values
genotypename	String	input genotype file	
snpname	String	input snp file	
outputformat	String	Can be one of the following: ANCESTRYMAP, EIGENSTRAT, PED,	

		PACKEDPED or PACKEDANCESTRYMAP	
genotypeoutname	String	output genotype file	
snpoutname	String	output snp file	
indivoutname	String	output indiv file	
OPTIONAL PARAMETERS			
familynames	String	Only relevant if input format is PED or PACKEDPED.	
noxdata	Boolean	If set to YES, all SNPs on X chromosome are removed from the data set.	
nomalexhet	Boolean	If set to YES, any het genotypes on X chr for males are changed to missing data	
badsnpname	String	Specifies a list of SNPs which should be removed from the data set	
outputgroup	Boolean	Only relevant if outputformat is PED or PACKEDPED	NO

- familynames : If set to YES, then family ID will be concatenated to sample ID. This supports different individuals with different family ID but same sample ID. The convertf default for this parameter is YES.
- noxdata: The convertf default for this parameter is NO.
- nomalexhet: The convertf default for this parameter is NO.
- badsnpname: Same format as example.snp. Cannot be used if input is in PACKEDPED or PACKEDANCESTRYMAP format.
- outputgroup: This parameter specifies what the 6th column of information about each individual should be in the output. If outputgroup is set to NO (the default), the 6th column will be set to 1 for each Control and 2 for each Case, as specified in the input indiv file. [Individuals specified with some other label, such as a population group label, will be assumed to be controls and the 6th column will be set to 1.] If outputgroup is set to YES, the 6th column will be set to the exact label specified in the input indiv file. [This functionality preserves population group labels.]

10. Download & Installation

a) Download instructions for the program

Follow these steps to download and install ANCESTRYMAP on your computer:

1. Go to http://genepath.med.harvard.edu/~reich
2. Click on: Download for UNIX or Linux or MAC for the source code
3. Make sure to rename the downloaded file to ancestrymap.tar.gz, since the download process sometimes renames it otherwise.

4. Decompress the file using gzip –d ancestrymap.tar.gz, and you should now see ancestrymap.tar
5. Unarchive this file using tar –xvf ancestrymap.tar. This will create a directory called *ancestrymap,* and the following directory structure under it:
   examplesfiles/
   src/
   bin/
   README file

In the *examplefiles* directory we have the following files:

Parmeter Files:

- **paramfile**: In the format of parameter file for *ancestrymap,* with additional parameters that are new in Version 2.0
- **parmono**: In the format of parameter file for *cntmono*
- **param0, param1, param2:** Parameter files for *ancestrymap*, discussed later in this section
- **parsim:** Parameter file for *ancestrymap* when running simulations
- parsim2d: Parameter file for *ancestrymap* when running fine-mapping simulations
- paramped: Parameter file for *ancestrymap* when using PED files
- par:8001-par:8009 : Parameter files for fine-mapping runs as generated by running the script *mkfine*.
- Example files as used by *convertf* executable, look at the relevant documentation for details

Input Data Files:

- **indiv.dat**: individual input file for *ancestrymap*
- **indiv1.dat** : Individual file for *ancestrymap* with one samples set to Ignore
- **geno.dat**: genotype input file for *ancestrymap*
- **snpcnts**: marker input file for *ancestrymap*
- **badsnps**: input file for *ancestrymap* with markers that need to be removed from the analysis
- **snps**: marker input file for *cntmono* or *ancestrymap*
- **aflist, eurlist***:* Ancestry files for *cntmono*

The genotype and individual files in this directory were generated by running simulations, and the marker files correspond to data reported in the Smith et al paper.

Output Files:

- **out2.dat** Output file generated by running *ancestrymap* using paramfile
- **out0.dat**: Output file generated by running *ancestrymap* using param0
- **out1.dat**: Output file generated by running *ancestrymap* using param1
- outsim2d.dat : Output file generated by running *ancestrymap* using parsim2d
- **admckout.dat**: Ouput file generated by running *admcheck* on out1.dat
- **indjunk**: Output file with detailed individual data created by running ancestrymap using param0

- **snpjunk**: Output file with detailed marker data created by running ancestrymap using param0
- **Fine-mapping Run Output Files**:
    1. badlist1, framelist1: Output files generated by *mkfine* script
    2. badlist:8001-badlist:8009: Bad marker files generated by *mkfine* for each fine-mapping run
    3. xx:8001-xx:8009: Output file for each fine-mapping run
- **outfiles/:** This is a directory which contains the output files mentioned in Section 6, and ancestry estimates for various markers considered in the fine-mapping run (gams:8001-gams:8009).

bin/ has the following executables:

- *cntmono*
- *ancestrymap*
- *baseprog*
- *admcheck*: A perl script which is used to extract the top "bad" markers
- *mkfine:* A perl script which is used to kick-off the fine-mapping runs
- *parfine.temp* : A accompanying template parameter file needed by the *mkfine* script
- *addcol, uniqit:* Helper perl scripts needed by *mkfine*

src/ has the C source code for making the *ancestrymap, cntmono, baseprog, convertf* executables, the library nicklib.a and a makefile called Makefile. The makefile can be used to make just the individual executables, or just the library or all together. This also has the following directories:

- *smartinclude/* has header files which are needed by the source code, users should not delete these files to ensure proper compilation of the code.
- *smarttables/* is needed by the source code.
- *nicksrc/: nicklib* source code

b) Running the example files

There are a number of parameter and corresponding output files in the *examplefiles/* directory. We recommend the user going through the following steps with these files before running their own data on ANCESTRYMAP.

- First step makes sure that the input files are in the right format. In this step we look at the parameter file parbaseprog and it's corresponding output file outbaseprog.dat. To run this type on the command line in the examples directory:

```
>> ./ancestrymap –p parbaseprog > outbaseprog.dat&
```

If there is any problem with any of the input files, one will see an appropriate message in the output file.

- Next step performs a couple of data checks. In this step we look at the parameter file param0, and its corresponding output file out0.dat. The key

parameter values in this file are *numburn* = 0, *numiter*s= 0, *checkit* = YES and *details* = YES.

To run this parameter file type on the command line in the *examples* directory:

>> ./ancestrymap –p param0 > outp0&

Compare the output files out0.dat (in the *examples* directory) and outp0 to make sure that you can understand the output generated. Note that the use of the random number generator makes it impossible for the results to be exactly the same for two runs unless the parameter *seed* has the same value.

Next, look at the output file indjunk generated by this run. From this file one can extract a list of individuals with very small number of genotypes by sorting it by the Num_valid_genotypes column. We will set the Status field to Ignore for some of these individuals in a copy of the original individual file called indiv1.dat file, since a lot of missing data will cause ANCESTRYMAP to behave badly. Also, one should discard markers which have low parental genotype counts by looking at the file snpjunk which can be done by looking at the fields PopA_vart, PopA_ref, PopB_vart and PopB_ref in this file. The discarded markers can be put in the "badsnpname" file. Next look at the output file out0.dat, where the checkdup and fastdup programs have flagged a number of duplicate individuals. We shall set the Status field to Ignore for one of these pair of individuals in the indiv1.dat file as well.

Thus the key focus in this step is to ensure that ANCESTRYMAP can successfully process the input files, and the identification of individuals which are duplicates or have very few genotypes, and markers with low parental genotype counts.

- The next step involves running a lot of data checking programs. In this step we will look at the parameter file param1, and its corresponding output file out1.dat. The key parameter values in this file are *numburn* = 5, *numiters* = 5, *checkit* = YES and *details* = YES. This corresponds to having very few burn-in or follow-on iterations and sets up ANCESTRYMAP in the mode to run the various data checking programs.

To run this parameter file type on the command line in the *examples* directory:

>>./ancestrymap –p param1 > outp1&

Compare the output files out1.dat (in the *examples* directory) and outp1 to make sure you can understand the various output sections. Note that the use of the random number generator makes it impossible for the output to be exactly the same for two runs unless the parameter *seed* has the same value.

Note that in the output file there are results from a large number of data checking programs. To extract the top markers that have failed the various checks run the perl script admcheck by typing on the command line:

>>admcheck out1.dat > ancsycheck.dat&

Compare the file ancsycheck.dat with the file admchkout.dat in the examples directory.

Here is an example of the output generated by *admcheck* and pointers on how to extract the bad markers.

From the *ancsycheck.dat* file we will pick the markers that are outliers for the various checks, and will add them to our *badsnpname* file which will allow the software to ignore these markers for the rest of the analysis. In addition, the user must also add to this file one of the pairs of markers which are in strong linkage disequilibrium with each other. It is necessary to remove these markers since otherwise one will see spurious results. Note that since we don't really have any bad markers, the badsnps file in the *examples* directory is just a sample file.

- Next we will look at the parameter file param2, and its corresponding output file out2.dat. This file corresponds to having 50 burn-in and 100 follow-on iterations, with *checkit* = NO, *details* = YES and uses the badsnps file that we created in the previous step.

To do this type on the command line in the *examples* directory:

```
>> ./ancestrymap –p paramfile > outf&
```

Compare the output files *outf* and *out2.dat* to make sure you can understand the output generated. Note that the use of the random number generator makes it impossible for the results to be exactly the same for two runs unless the parameter *seed* has the same value. The important things to focus on in this run are the τ(Afr) and τ(Eur) values, scores for the various chromosomes and the genome log factor value.

In addition to the standard output, this parameter file will also create a number of output files in the *outfiles* directory. These files are as follows, and have been discussed in detail in the documentation.

- act.out
- freq.out
- snp.out
- theta.out
- lambda.out
- ethinc.out
- ind.out

c) Running the program with user data

To run the executable with user specific data, create the input files in the format specified in Section 4 (also see sample files in the *examples/* directory). Then follow these steps:

1. Make sure that the program can read all the input files successfully, by typing on the command line:

   ```
   >> ./ancestrymap –p parbprog. > outbaseprog.dat
   ```

   Here *parbprog* should be made using the file *parbaseprog* as a sample.

2. Run *ancestrymap* using the parameter file param0 as a sample file.
   >> ./ancestrymap –p parc0. > out0.dat
   Here *parc0* should be made using the file *param0* as a sample, and corresponds to setting *checkit, details* = YES, and *numburn, numiters* = 0. From the file corresponding to the parameter *indoutfilename*, get a list of individuals which have very few genotypes and set their Status field to Ignore in the individual file. From the file corresponding to the parameter *snpoutfilename* get a list of markers which have low parental genotype counts, and put them in the file corresponding to the parameter *badsnpname*. Next from the output file generated by this run extract the pair of duplicate individuals (if any), and set the Status field to Ignore in the individual file for one of the pair of individuals.

3. Run *ancestrymap* using the parameter file param1 as a sample file. This forces various data checking algorithms mentioned in Section 5 to run, and corresponds to *checkit, details* = YES, and *numburn, numiters* = 5.
   >> ./ancestrymap –p parc1 > out.dat.

4. To extract the top markers which have failed the various tests run:
   >> admcheck out.dat.
   admcheck extracts a list of the top 10 markers with the highest scores for hwcheck, mapcheck and freqcheck ; and individuals with highest and lowest scores for checkindiv (See Section 5). From this list the user should choose the markers to add to their "badsnps" file, using the guidelines offered in the documentation, and the example output. One should also include one of the pair of markers that are in linkage disequilibrium with each other in the badsnps file. Also, one should set the Status field to Ignore in the individual file for individuals which fail the checkindiv test, using the guidelines in Section 5.

5. Run *ancestrymap* using the parameter file paramfile as an example, using a larger number of both burn-in and follow-on iterations and with *checkit* = NO. Make sure to include the badsnps file in parameter file, to ensure that the software does not include them in the analysis any more.

d) Building your own executable from source code

- The first step is to go to the */ancestrymap/src/* directory.
- Next type on the command line:
  >> make all
- This will make the library *nicklib.a* and the executables *baseprog, convertf* and *ancestrymap*

11. Simulations

In order to perform simulations one must set *dotoysim* = YES. Some of the other parameters that can be set are *casecontrol*, *markersim*, *risksim, simnumindivs*. Default and

possible values for these parameters are discussed in Section 3. An example simulation parameter file is included in the *examples* directory and is called *parsim*. One can perform simulations under a variety of scenarios, as outlined in the paper. One also perform simulations in the fine-mapping mode, some of the parameters that can be set are sim2dvals and sim2d_caseonly.

12. How to cite this program

To cite this program refer to it as ANCESTRYMAP, and give the reference to the accompanying paper by Patterson et al paper.1

13. Tutorial

Go to the website http://genepath.med.harvard.edu/~reich/Tutorial.htm for an online tutorial.

14. Description of the auxiliary package : getpars, cntmono

a) Cntmono Program

i) Overview

In this section we will discuss the auxiliary software cntmono included with ANCESTRYMAP. This program takes as input, files with genotype data, and corresponding individual and marker data for the modern counterparts of the ancestral parental population subgroups and creates an output file which has the counts for the reference and variant alleles for these populations. This can in turn be used by the *ancestrymap* program as its input marker/ snp file. For a short tutorial on how to run this program click here.

ii) How to run the program

The command line arguments for *cntmono* are p (parameter file) which is mandatory, V (version number) and v (verbose mode), same as for *ancestrymap*. To run this type on the commandline:

>> cntmono –p parcmono or ./cntmono –p parcmono

Use the file parmono in the *examples/* directory as a sample file to create parcmono. This file has names for the following files:

- **Genotype Data**: String: *genotypename*

SNP_ID	INDIV_ID	Vart_allele_cnt
rs112	I1	1
rs113	I2	0
rs114	I3	2

- **Marker Data**: String: *snpname*

SNP_ID	Chr_Num	Gen_Pos	Phys_Pos
rs112	1	0.3455676	114556
rs113	2	0.566879	1400898

- **Individual Data**: String: *indivname*

INDIV_ID	Gender	Status
I1	F	CEPH
I2	M	BOTSWANA
I3	M	GHANA

The value in the Status field is that of the ethnicity of the sample. This can be as varied as the examples shown above, or as narrow as AFRICAN, EUROPEAN, ASIAN, etc.

- **Population A subgroup data**: String: *aglistname*

This contains a list of ethnicities that are classified as population A subtype

Ethnicity
Beni
Botswana

- **Population B subgroup data**: String: *eglistname*

This contains a list of ethnicities that are classified as population B subtype

Ethnicity
CEPH
Italy

- **Output Data**: String: *output*

The output file has the following columns

- SNP_ID
- Chromosome number
- Genetic position
- Physical position
- Population A reference allele count
- Population A variant allele count
- Population B reference allele count
- Population B variant allele count
- Number of valid genotypes

Before using the output file generated by *cntmono* as the input marker file for *ancestrymap,* remove from it the blank lines, lines with comments and, the header line. This file should only contain details about the markers, else *ancestrymap* will give a fatal error.

b) Getpars Overview

In this section we will discuss another auxiliary package getpars. This is a simple package for inputting parameters by keyword**.** It is crucial to reading and interpreting the parameter file, and to storing the various parameters specified in it. The package uses a "handle" (object) to store parameter information, and provides a simple string replacement facility. Parameters may appear in any order in the parameter file. Multiple parameter files can be dealt with, though the need for this would be unusual.

We will briefly outline the various functions that are part of this package:
1. *phandle *openpars(char *fname)* : Opens handle, and must be called before all other routines.Sample call: ph = openpars(myparameterfilename) ; In all of the following parameter setting routines, if the parameter is not present, the variable to receive the value will not be changed. Thus default values should be set before the call.

2. *int getstring(phandle *pp, char *parname, char **kret)* ;
Sample call: getstring(ph, "inputfilename:", &inputfilename) ;
Return value: positive integer if *parname* is found (and therefore *kret* is set) else negative integer. This is the same for all the analogous routines discussed below.

3. *int getint(phandle *pp, char *parname, int *kret)* : Sets an integer value. YES or NO values can be interpreted as 1, 0 respectively. This is convenient for setting boolean switches using "c" ints.
Sample call: getint(ph, "iterations:", &iterations) ;

4. *int getints(phandle *pp, char *parname, int *aint, int nint*) : Sets nint integer values into array aint. The values can be separated by white space or ':'.

5. *int getintss(phandle *pp, char *parname, int *aint, int *xint)* : Sets variable number of integer values into aint. Number set is returned in xint.

6. *void *closepars(phandle *pp)* : This is a destructor,and is called when parameter cracking is complete. All memory associated with ph is freed.

7. *void dostrsub(phandle *pp)* : We use the convention to insist that parameter names are lower case alphanumeric, and contain no upper case parameters. Now upper case "parameters" can be used for string replacement.
Example: parameter file mypars contains:
HOME: /home/harvey01/nickp
DIR: HOME/datadir
datafile: DIR/mydata
The following code fragment would be appropriate:
ph = openpars("mypars") ;

```
dostrsub(ph) ;
getstring(ph, "datafile"; &datafilename) ;
closepars(ph) ;
```

8. *void writepars(phandle *pp)* : writes a copy of the parameter file (after string replacement if dostrsub has been called) to standard out.

15. Troubleshooting & Bugs

16. Bibliography

i.Patterson et al (2004) Methods for High-Density Admixture Mapping of Disease Genes Am. J. Hum Genet. 74: 979-1000

ii. Smith et al. (2004) A High-Density Admixture Map for Disease Gene Discovery in African Americans Am J Hum Genet 74: 1001-1013

17. Contact Information

Nick Patterson: nickp@broad.mit.edu (617 252 7043)

David Reich reich@broad.mit.edu (617 432 6548)

Arti Tandon atandon@broad.mit.edu (617 432 5348). This is the person responsible for maintaining the website, and should be the one contacted in case of any issues with the software.

18. Appendix A: Expert use Parameters

Here is a table of the parameters that are for expert use only

Name	Type	Description	Possible and Default values
unknowngender	String	Unknown gender set to value	U
twomodels	Int	The two models A and B are defined with model A as the first twomodel number of risk values, and the rest being model B, scores are produced for each SNP for both models A & B, and genome wide stats and comparisons are made. The initial motivation was to do a kind of case-control	Positive integer Default: 0

		analysis with model A risk greater than 1 just for cases, and model B with same risk for both cases and controls	
theta2mode	Boolean	If YES, allows the θ value to be different on the two chromosomes. Used in cases where mother and father have very different ancestries.	0,1 Default: 0
lambda2mode	Boolean	If YES, allows the λ value to be different on the two chromosomes. Used when mother and father have very different ancestries.	0,1 Default: 0
keeptrashfiles	Boolean	Keep the α, β and g values from the last iteration	0,1 Default: 0
numlagprint	Int	Auto correlation statistics upto *numlagprint*	Positive integer >= 0 Default: 10
markerpub	Int	Used in debugging	Default: -1
allele_scale_fac	Double	Multiply allele counts by the scale factor	Default: -1
nopopsmode	Int	If YES, uses the initial frequency estimate to be the modern frequency counts, and then these modern frequency counts are not used	0,1 Default: 0
muval	Double	Used in bridge sampler	Default: 0
a1	Double	Used in simulation mode	
psi1	Double	Used in simulation mode	
p1	Double	Used in simulation mode	
B1	Double	Used in simulation mode	

19. Appendix B: Parameters new in Version 2.0

Parameter Name	Data type	Description	Possible and Default values
leave1mode	Boolean:	In checkit mode, leaves 1 marker at a time and gives the score	NO
dupmode	Boolean	In checkit mode, runs duplicate check on all indivs	NO
fastdup	Boolean	In checkit mode, runs fast duplicate check on all indivs. This parameter is automatically set to YES, in checkit mode	YES
usephyspos	Boolean	Calculates and resets the genetic positions based on physical positions, particularly useful when you don't have genetic map	NO
dumpgammas	Boolean	Dump gammas for all markers and indivs in binary format (will need another program to print in ASCII format: Alkes)	NO
gammafiles	String	This is the set of files which are created when using dumpgammas	NULL
emiter	Integer	This is the initial EM algorithm to initialize the MCMC	30
alldata	Boolean	Used in simulation mode, make simulated data for all the markers, if NO it keeps the	NO

		missing data intact	
lmmodel	Boolean	Used to run fine-mapping. This assumes ancestry risk is specified by risk model, actually 1D model	NO
lmchrom	Integer	The chromosome on which you run fine-mapping	
lmnumx	Integer	Number of points in allele risk mesh	Positive values > 1, default: 30
lmmax	Integer	Max value for the mesh, with number of mesh points being lmnumx	3.0
lmthresh	Double	If fine-mapping log score is below lmthresh, don't print details of fine-mapping runs	-10.0 (This is to get all the markers)
lmdetails	Boolean	Detailed output	YES
lmlobase	Integer	Physical position low end for fine-mapping	
lmhibase	Integer	Physical position high end for fine-mapping	
pubxindname	String	Name of a individual for which we want to print some output	
pubx	Integer Array	The first component is the internal individual ID for whom you want to output gammas	
pubxa	Integer Array	List of individual IDs for which we want to output gamma?	
markername	String	This is used to publish the gammas as well as in simulation mode if markersim is not specified	NULL

sim2dvals	Double Array of size 4	Specifies ancestry and allelic risk for fine-mapping simulations	Sim2dvals[0]: Afr freq Sim2dvals[1]: Caucasian freq Sim2dvals[2]: λ Sim2dvals[3] : μ
sim2d_caseonly	Boolean	2D simulation where we only consider the case genotypes?	NO
familynames	Boolean	Used with PED file, if we have unique Indiv Ids set this parameter to NO, else to YES	
indivoutname	String	Output file for individual information	
snpoutname	String	Output file for marker information	
genotypeoutname	String	Output file for genotype data	
localoutfilename	String	Output file with all the detailed information for markers	
mincasenum	Integer	Removes cases which have below a certain threshold # of genotypes, used in conjunction with the file badpairsname	1
casecontrol	Integer Array	Number of cases and controls, used in simulation mode	
hiclip	Double	Allows LOD scores to be upto a maximum of hiclip. Use if scores appear to be saturating	Default: 15.0
loclip	Double		Default: -20.0
packmode	Integer	This packs the genotype data. To be used if one has a memory extensive job	Default: NO

[1] Patterson et al (2004) Methods for High-Density Admixture Mapping of Disease Genes Am. J. Hum Genet. 74: 979-1000

[2] Smith et al. (2004) A High-Density Admixture Map for Disease Gene Discovery in African Americans

Am J Hum Genet 74 : 1001-1013