Evaluation of *vGWAS*, a test for determining scenarios of *epistasis*

Yonatan Ayalew Mekonnen

NORWEGIAN UNIVERSITY OF LIFE SCIENCES

Department of animal and aquacultural sciences (IHA)

Master Thesis 30 credits 2013

Evaluation of *vGWAS*, a test for determining scenarios of *epistasis*

by

Yonatan Ayalew Mekonnen

Master's Degree Project

TABLE OF CONTENTS Page

ABSTRACT:

A number of software packages have been developed for the detection of *epistatic* QTL. Nevertheless, none of these approaches could assure the optimal detection of *epistasis*. There are a number of limitations in these studies including high computational demands due to increased combinations of interactions for large datasets. Therefore, it has been a challenge to identify genes which are responsible for complex genetic traits caused by *epistasis* as well as gene by environment (G×E) interactions. Although genetic association studies have recently gained popularity, most of these studies have been carried out on the effect of the markers' mean trait value without considering the differences in the variances of the traits. These differences could be associated with detecting variance controlling loci and missing heritability. In this paper, we address the novelty of variance genome wide association study (*vGWAS*) and evaluate its power for the detection of *epistasis*. Data were generated using simulation program for two chromosomes, both containing 209 markers. *vGWAS* – an R statistical package was used with two and three loci *epistasis* model to analyze the simulated data. The Brown-Forsythe (Levene's) test was used to assess the equality of group variance for variance controlling loci. For both two and three loci *epistasis* models, correct QTL are detected. Both heritability and power are calculated using three genotype-phenotype maps. Using *vGWAS* is therefore one way to find some of the variation currently unexplained. Interaction variances (V_I), which are included in *vGWAS* analysis, is a novelty compared to other software in terms of detecting variance controlling loci (vQTL) and finding the missing heritability due to interacting loci. In our analysis, we have shown that at low or moderate noise level, both power and heritability estimates were approximately 1. The result reflects that *vGWAS* is a powerful and efficient tool for detecting candidate *epistatic* genes (*i.e.*, detecting vQTL are often associated with interacting genes). Therefore, it could be appropriate to be used as a procedure for real data dealing with complex disease or obscured genetic interactions in addition to standard linear regression models.

Keywords: *Epistasis*, heritability, interaction variances, power, QTL, *vGWAS*, vQTL

ABBREVIATIONS

ANOVA	Analysis of variance
BQTL	Bayesian quantitative trait mapping
cM	Centimorgan
COE	Convex optimization based *epistasis* detection algorithm
FPR	False positive rate
G×E	Gene-environment interaction
G×G	Gene-gene interaction
GWAS	Genome wide association study
H^2	Heritability
MDR	Multifactor dimensionality reduction
MQTL	Multi-environmental QTL analysis
QTL	Quantitative trait locus
QTLBIM	Quantitative trait locus Bayesian interval mapping
SNP	Single nucleotide polymorphism
TEAM	Tree based *epistasis* association mapping
V_A	Additive variance
V_D	Dominance variance
V_E	Environmental variance
V_G	Genetic variance
vGWAS	Variance genome wide association study
V_I	Interaction variance
V_M	Mean difference
vQTL	Variance controlling loci
V_V	Variance difference

1. INTRODUCTION:

The term ‘*epistasis*’ was first coined by Bateson in 1909 to explain the masking effect of alleles at one locus to prevent an activity at another locus (Cordell *et al*., 2001). Bateson used the term ‘*epistasis*’ to describe the phenomena that a novel phenotype results when alleles are combined compared to when they are apart. This concept was considered as an extension of dominance notion at single locus. Fisher (1918) used the term *epistacy* to describe the interaction between alleles at different loci (*i.e.*, deviation from additivity), which contribute to a specific phenotype or trait (Malmberg and Mauricio, 2005). However, a more general term ‘gene interaction’ was used by several researchers to describe the concept *epistasis*. From these point of view, both Fisher’s and Bateson’s definitions of *epistasis* can be used to explain gene interactions at various levels (Phillips, 2008). Carlborg and Haley (2004) stated that *epistasis* is the interaction between loci in which the phenotype depends on the genotype of one locus in the context of the second locus. Phillips (2008) summarizes the various views of *epistasis* as functional, compositional and statistical *epistasis.*

Functional epistasis deals with the interaction of proteins and other genetic elements at molecular level. For instance, functional relationship disorder between proteins involves interaction without direct genetic involvement although the cause of the disorder has genetic basis. Protein-protein interaction is used to address *epistasis* of this type (Phillips, 2008).

Compositional epistasis is used to describe the blocking effect of one allele to another which reflects classical definition of *epistasis*. Substituting one allele at the loci of interest without changing the background genes will influence the effect of a specific set of alleles in another locus (Phillips, 2008).

Statistical epistasis, which reflects the Fisherian view, rather considers the average deviation of alleles from the additive combination at different loci. The views of compositional and statistical *epistasis* seem slightly contradicting. Compositional *epistasis* evaluates the effect of allele substitution against fixed background genes whereas statistical *epistasis* measures the average allele substitution effect against average genetic background of the population. Therefore, allelic substitution under various genetic backgrounds (*i.e.*, either fixed or average) unifies these two approaches of *epistasis* (Phillips, 2008). However, Phillips definition of functional *epistasis* is not the same as Alvarez-Castro and Carlborg's definition of functional *epistasis* (Cordell, 2009). In their unified model, the structure of algebraic formulation of functional *epistasis* resembles that of statistical *epistasis*. But, instead of using average allelic substitution effect, their natural and orthogonal interaction model (NOIA) uses natural (non-average) effect of allelic substitution for functional *epistasis* formulation (Alvarez-Castro and Carlborg, 2007). The term biological, genetical and physiological *epistasis* has been interchangeably used to address functional *epistasis* in the literature, though (Moore and Williams, 2005).

On the other hand, quantitative geneticists consider *epistasis* in terms of additive and dominance interactions. Lynch and Walsh (1998) described dominance and *epistasis* as two measures of non additivity in which the former explains allelic effect within locus whereas the latter explains allelic effects between loci. Without considering *epistasis*, the total genetic variance of a locus is simplified as additive and dominance effect (Lynch and Walsh, 1998).

$$\sigma_G^2 = \sigma_A^2 + \sigma_D^2 \qquad [1]$$

Where, σ_G^2 is the total genetic variance, σ_A^2 is additive genetic variance and σ_D^2 is dominance genetic variance.

For two interacting loci, for instance, *epistasis* can arise in three different ways: additive x additive (αα), additive x dominance (αδ) and dominance x dominance (δδ). Similarly, for interacting three loci, there are four different ways in which *epistasis* can arise: additive x additive x additive (ααα), additive x additive x dominance (ααδ), additive x dominance x dominance (αδδ), dominance x dominance x dominance (δδδ) and the number increases with increasing interacting loci.

$$\sigma_G^2 = \sigma_A^2 + \sigma_D^2 + \sigma_{AA}^2 + \sigma_{AD}^2 + \sigma_{DD}^2 + \sigma_{AAA}^2 + \sigma_{AAD}^2 + \sigma_{ADD}^2 + \sigma_{DDD}^2 \ldots \qquad [2]$$

This means that *epistatic* interaction can influence on the additive and/or dominance components of genetic variance. Although the individual epistatic effects of loci are small, the sum of these individual effects may be large. This may persuade quantitative geneticists not to ignore *epistasis*.

However; Crow (2010) proclaimed that it was reasonable to ignore *epistasis* in some circumstances in the prediction. Since quantitative geneticists consider quantitative phenotypes rather than individual genes' effects, variance due to *epistasis* would have small effect on predicting breeding values for selection of individuals. According to breeders, many genes which contribute to quantitative traits usually have little dominance or *epistasis*. The reason for this is that each continuously distributed quantitative trait would have a small contribution to quantitative measurements and their small effects are given as additive variance (Crow, 2010). According to Crow and Kimura (2009), breeders only account for additive variances although there may be large amount of *epistasis* and dominance. They regarded the effect of *epistasis* as a noise or a complex factor obscuring selection progress. However, Maher (2008) points out that these small effects of individual and cumulative sets of genes could be associated with genetics of common disease and missing heritability. He postulated that one source of the missing heritability, often encountered in genome wide association studies (GWAS) could be due to *epistasis*. However, there have been limitations to evaluate their quantitative significance. These limitations were lack of good statistical power and proper experimental design.

In this manuscript, we investigate variance genome wide association study (*vGWAS*), complementary to the current models that use differences in phenotypic variances between genotypes rather than using mean differences to identify the loci of interest (Shen *et al*., 2011). The aim of this paper is to evaluate the power of *vGWAS* in relation to broad sense heritability for the detection of candidate epistatic loci. The Brown-Forsythe test for genotypic heteroscedasticity is used in *vGWAS*. The paper also addresses the novelty of *vGWAS* in detection of *epistasis* as a procedure to be used in addition to standard linear regression models. Moreover, this thesis will intend to fill the gap for animal breeding and genetics by providing more powerful method in terms of detecting interacting loci that can be used for animal selection in particular and for complex disease genetics study in general. However, there were limitations to get adequate references related with the topic.

1.1 The development of QTL mapping for detecting *epistasis*:

In quantitative trait loci (QTL) mapping, the population should be partitioned into different genotypic classes. Then, the applied test statistic should confer whether the individual of one genotype differ significantly from individual of other genotype with respect to a certain phenotype. Usually, QTL mapping requires segregating mapping population such as F_2 cross populations, backcross populations, recombinant inbreed lines, near isogenic lines or double haploids lines. Phenotypic data obtained from backcross or intercrosses are used to identify the genomic region that has genotype-phenotype association. In order to identify the genomic regions associated with the trait of interest, genetic markers such as microsatellite and SNPs; phenotypes (*i.e.*, observed characters of the individuals) and genetic maps, which specifies the location of markers on the chromosome, are used (Wu *et al*., 2007). QTL linkage analysis and fine mapping studies could lead to identify and functionally confirm the candidate genes which have potential effect on the trait. For instance, both DGAT1 and GHR genes, which have significant effect on milk production trait in dairy cattle, have been identified and functionally confirmed using QTL mapping and fine scale mapping primarily (Jiang, 2010).

A number of methods and software to detect *epistasis* between QTL have been developed during the past years. For instance, Genetic algorithm by Carlborg *et al*. (2000); Exhaustive algorithms by Nelson *et al*. (2001) and Ritchie *et al*. (2001); Two-step approach by Storey *et al.* (2005) and Evans *et al*. (2006); Three *epistasis* detecting tools (*i.e.*, fast ANOVA, COE and TEAM) by Zhang *et al*. (2010). Currently available methods for estimating QTL parameters use least square regression, maximum likelihood and Bayesian regression as their main methods as shown in Table 1. All packages listed in Table 1 are limited to detect interaction between two loci without considering higher order interactions. This is because testing all pair wise combination creates computational burden for the analysis and becomes time consuming. Although the package PSEUDOMARKER uses DIRECT, which is a computationally efficient algorithm on MATLAB platform, the current application is only for two QTL scans. Some of these packages also have *epistatic* searching algorithms in addition to QTL detection.

Table 1. Existing software packages for QTL analysis.

Main methods	Packages
Least square regression	PSEUDOMARKER, QTL Cartographer, HAPPY, MapQTL, BQTL, QTL Network, WebQTL, MultiQTL, Map Manager QTX, PLABQTL, the QTL Café, GridQTL, R/qtl, PROC QTL, MQTL, QGene, QTL-All, Epistat
Maximum likelihood	Mapmaker/QTL, QTL Cartographer, MapQTL, BQTL, Multimapper, PROC QTL, QTL-by-SAS and QGene
Bayesian regression	QTLBIM, BQTL, Multimapper, PROC QTL, Shrinkage QTL and QGene
Epistatic searching algorithms	PSEUDOMARKER, QTLBIM, BQTL, QTLNetwork, MultiQTL, PLABQTL, GridQTL, R/qtl, PROC QTL, Map Manager QTX and Epistat

1.2 The development of genome wide association studies (GWAS) and other methods for detecting *epistasis*:

Along with the advent of single nucleotide polymorphisms (SNPs), genome wide association study becomes powerful to detect and identify the genomic regions harboring causal variants. For complex diseases such as autism and schizophrenia, the application of genome wide association studies is dominantly used. But a few applications of GWAS have been performed in cattle such as identifying genes for milk production traits (Jiang *et al.*, 2010). To perform GWAS, two groups are required; case and control groups with large number of data sets. The sample taken from each of the two groups is then scanned for intentionally selected markers (*i.e.*, using existing data such as HapMap) or randomly chosen markers (SNPs). If the result indicates that the genetic variations in the case group are considerably more frequent than the control groups, then the variations are strong indicators of the region in the genome where these variations are associated with a certain phenotype. However, in order to identify the exact genetic changes associated with a certain phenotype, further step is needed such as sequencing.

GWAS is denoted as a powerful method for the study of disease-associated genetic variants (Iles, 2008). Unlike QTL mapping, GWAS utilizes many unlinked markers; does not necessarily require inbred lines and is able to detect many more interacting genes (Corvin, 2010). Most of these studies have been carried out on the effect of the markers' mean trait value.

Until recently, the differences in the variances of these traits were not considered. Pare´ *et al*. (2010) proposed a novel method to prioritize markers for gene-gene and gene-environment interaction. The analysis was done in two steps. The first step was prioritizing markers for further interaction using Levene's equality of variance test (*i.e.,* assess whether k samples have equal variance). The second step was testing prioritized markers using linear regression for interaction effects against an environmental covariate or other markers. This technique opened a new perspective to weigh G×G and G×E interaction on which the new method (*vGWAS*) is based. There was some doubt about linking variance differences to *epistasis* until it was realized that the source of these variances could be *epistasis* or G×E interactions (Pare' *et al*., 2010). Quantitative traits, which are regulated by genetic mean effect, have been explored and were found by most of the genetic association studies. Unlike the mean controlling loci which contribute to additive genetic effect, variance controlling loci affect phenotypic variance indirectly. This means that other genetic or environmental factors affect the mean shift due to variance controlling loci rather than directly contributing to additive genetic variance. Variance controlling genes are therefore important for the genetic robustness by stabilizing the traits that are under selection. Integrating mean controlling loci, variance controlling loci and loci controlling both are therefore important to elucidate the genetic architecture of a given trait (Shen *et al.,* 2011).

For the detection of higher order interactions in association studies, a new method called multifactor dimensionality reduction (MDR) was developed by Ritchie *et al*., (2001). Most importantly, this method identifies multi-locus interactions, rather than only single locus effect. This makes it possible to determine the mechanisms of disease susceptibility that underlie the influence of *epistasis* by looking at the hierarchy of interacting genes in biological networks (Moore, 2003). Most commonly, logistic regression model, which includes main and interaction terms, is used to test statistical interactions. The whole-genome analysis package PLINK is endowed with logistic regression tests for both main effect and the interactions (Cordell, 2009). However, logistic regression has low power when loci have no marginal effect. Therefore, the method which is developed by Yang *et al*. (1999) can be used to compare case and controls for their inter-locus associations based on partitioning of X^2 values for the conditions that logistic regression has low power. Although several efforts have been made to study interactions alone, methods that allow interactions with other genetic or environmental factors while testing the effect of a given locus increase the power (Cordell, 2009).

Analyzing and testing all possible pairs of loci is the simplest way to look for interactions. Exhaustive search were built to analyze data in this way. If exhaustive search is applied to genome scan, it might take several hours or days or even a month to make all pair wise comparisons (Cordell, 2009). But for higher-order interactions such as three loci, four loci or higher level interactions, exhaustive search are not normally used for the analysis due to high computational burden. In order to overcome this problem, two-step approaches have been used. In the first stage, the loci that are significant for a certain threshold level are filtered. Secondly, an exhaustive search on those selected loci is applied for two locus or higher order interactions (Hoh *et al*., 2000). This approach allows loci for the subsequent stage of testing if they had marginal association with the trait. Therefore, a shortcoming of this method is that alleles which did not show marginal associations are not evaluated to detect interactions. Two step methods are not the only methods to overcome high computational burden. Alternatively, most of machine-learning or data mining approaches such as Relief and Random Forest do not necessarily require a locus with marginal effect (Cordell, 2009). Although an efficient global optimization algorithm, DIRECT, was developed to reduce the high computational complexity demand (Ljungberg *et al*., 2004), scaling problem remains a challenge for genotype-phenotype mapping due to exponentially increasing numbers of all possible genetic interactions (Phillips, 2008).

1.3 Limitations to detect *epistasis*:

It has been a challenge to identify genes which are responsible for complex genetic traits especially when they are involved in *epistasis* and gene by environment (G×E) interactions. It is also clear that gene frequencies and phenotypic variations change from population to population, from sample to sample and from generation to generation. This makes it difficult to assign a phenotype to its complimentary genotype (even when the system is simple and there are no environmental interactions). Since there is often some degree of environmental interaction, it becomes more difficult to detect and map genes due to non-linear interactions between genes. As the number of genetic factors increases, the average contribution of each factor explaining a specific phenotype decreases. The inverse relationship between number of genes and individual average gene effect limits detection and mapping of genes. Caution is therefore important when we want to map the component of genes in complex traits and explain their role (Wade *et al*., 2001).

Since *epistatic* QTL uses the mean of multi-locus genotypes rather than single locus individuals, it requires larger sample size. Usually, there is a limitation to collect adequate data in association studies which results in lack of power (Carlborg and Haley, 2004). Most studies for detection of epistatic QTL use quantitative genetic models since the detected interactions are not always biologically relevant. For instance, when *epistasis* is modeled as a deviation from additivity, it is difficult to report gene-gene interaction in a biological context (Cordell *et al*., 2001). But there are methods to associate statistically modeled *epistasis* to their real biological meanings. One way to describe how gene interactions influence the phenotype is genotype-phenotype maps. These maps can be used to connect experimental data to real gene interaction patterns (Carlborg *et al*., 2003). Functional relationship among loci and gene regulatory networks such as positive and negative feedback loops are other methods to link statistical estimates of *epistasis* to biological meaning (Omholt *et al*., 2000).

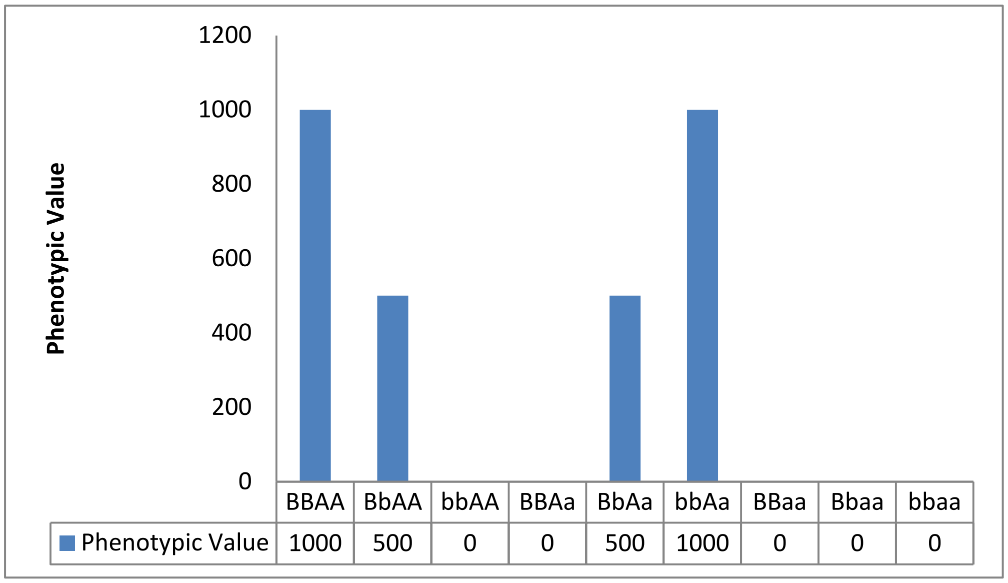
1.4 The missing heritability

The majority of complex diseases are thought to be influenced by many environmental and genetic factors (Manolio *et al*., 2009). GWAS has become a powerful and popular tool to associate the genotypic variance with phenotypic variance. Complex traits are however often assumed to follow the infinitesimal model (*i.e.*, traits are determined by infinite number of unlinked loci) (Fisher, 1918). Several other reasons, including larger number of variants with smaller effect or few variants with larger effect and a low power to detect gene-gene interaction, may lead to the small estimates of heritability observed in many studies. This raises a question: Is the "missing heritability" due to sampling issues or an inherent problem with the methodology in association studies? It was termed as "dark matter" of the genome, since it certainly exists, but cannot be detected. Therefore, there is a need to propose and develop methods to detect the potential source of this missing heritability when dealing with complex disease genetics (Manolio *et al*., 2009). Since current methods used in GWAS are not able to detect interactions, using *vGWAS* enables to find some of the variation currently unexplained.

2. MATERIALS AND METHODS:

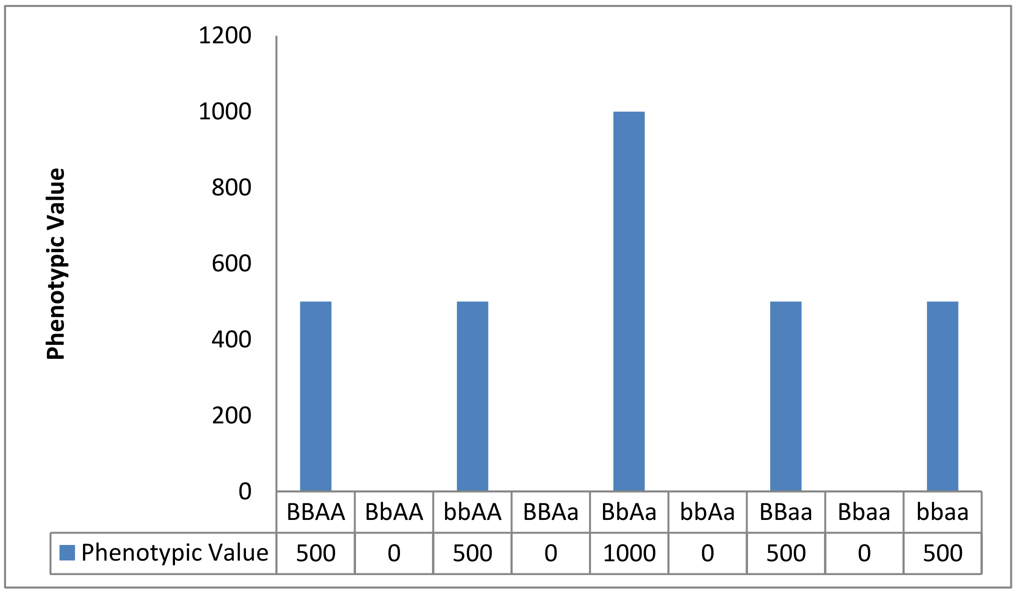
2.1 Genotype-phenotype map creation:

For two and three loci *epistasis* model, different genotype-phenotype maps were used. A genotype-phenotype map without mean difference could inherit variance difference between genotypes of a particular individual locus. Since *vGWAS* was built to detect variance differences, QTL of these maps can be detected using this program. Genotype-phenotype maps (shown in Figure 1a, 1b and 1c) were used to show variance differences between genotypes and estimate the power of *vGWAS* in relation to heritability.



	BBAA	BbAA	bbAA	BBAa	BbAa	bbAa	BBaa	Bbaa	bbaa
🟦 Phenotypic Value	1000	500	0	0	500	1000	0	0	0

Figure 1a. Genotype-phenotype map. *There is no mean or variance difference between genotype AA and Aa, when summed over all combination with the B-locus. But, there is mean and variance difference between genotype aa and the other genotypes (AA or Aa) averaged over all combination with the B-locus.*



	BBAA	BbAA	bbAA	BBAa	BbAa	bbAa	BBaa	Bbaa	bbaa
🟦 Phenotypic Value	500	0	500	0	1000	0	500	0	500

Figure 1b. Genotype-phenotype map (including without mean difference for genotype-phenotype maps having same frequencies). *There is no mean difference between genotypes AA, and Aa, aa and AA and; aa and Aa when summed over all combination with the B-locus. But there are variance differences between genotype AA and Aa; and Aa and aa, averaged over all combination with the B-locus.*

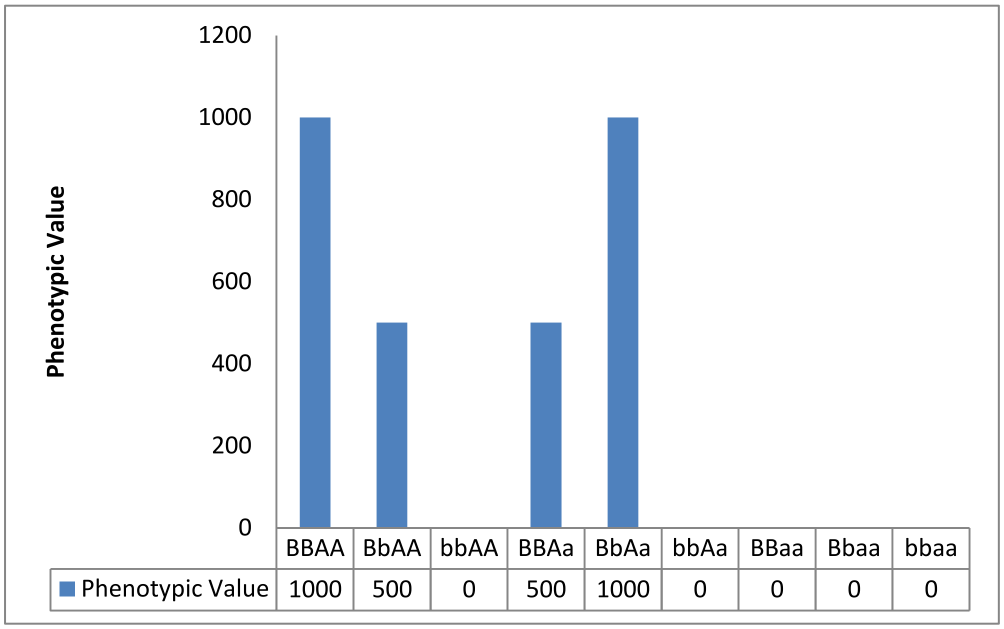


Figure 1c. Genotype-phenotype map. *There is no mean and variance difference between genotype AA and Aa, when summed over all combination with the B-locus. There is mean and variance difference between genotype AA and aa, and Aa and aa averaged over all combination with the B-locus.*

The numbers in each cell represent the mean phenotypic value of specific genotype. Since the population is in Hardy-Weinberg equilibrium, there may be a small main effect. This means that the differences observed could be detected by testing for main effect. However, testing the variance should not be greatly affected by these artifacts. In non Hardy-Weinberg population, there may be a condition in which the genotype frequencies remain the same. In this case, *vGWAS* could rather detect differences in variance without main effect.

2.2 Data simulation:

Data was generated up to 30 generations using a simulation program written by Ronald Nelson in C++ and compiled for Mac_OS_X (see appendix). Simulations were conducted for different scenarios of population size and noise levels at generation 30 as summarized in Table 2. To limit the effect of drift on the allelic frequency, we terminated at generation 30. This could ensure enough recombination events to be similar to a realistic out-bred population. Two chromosomes were simulated; the first contains 100 markers and the second 109 as shown in Figure 2. QTL are simulated in the genome using these two chromosomes and keeping them far apart to behave as if they are unlinked.

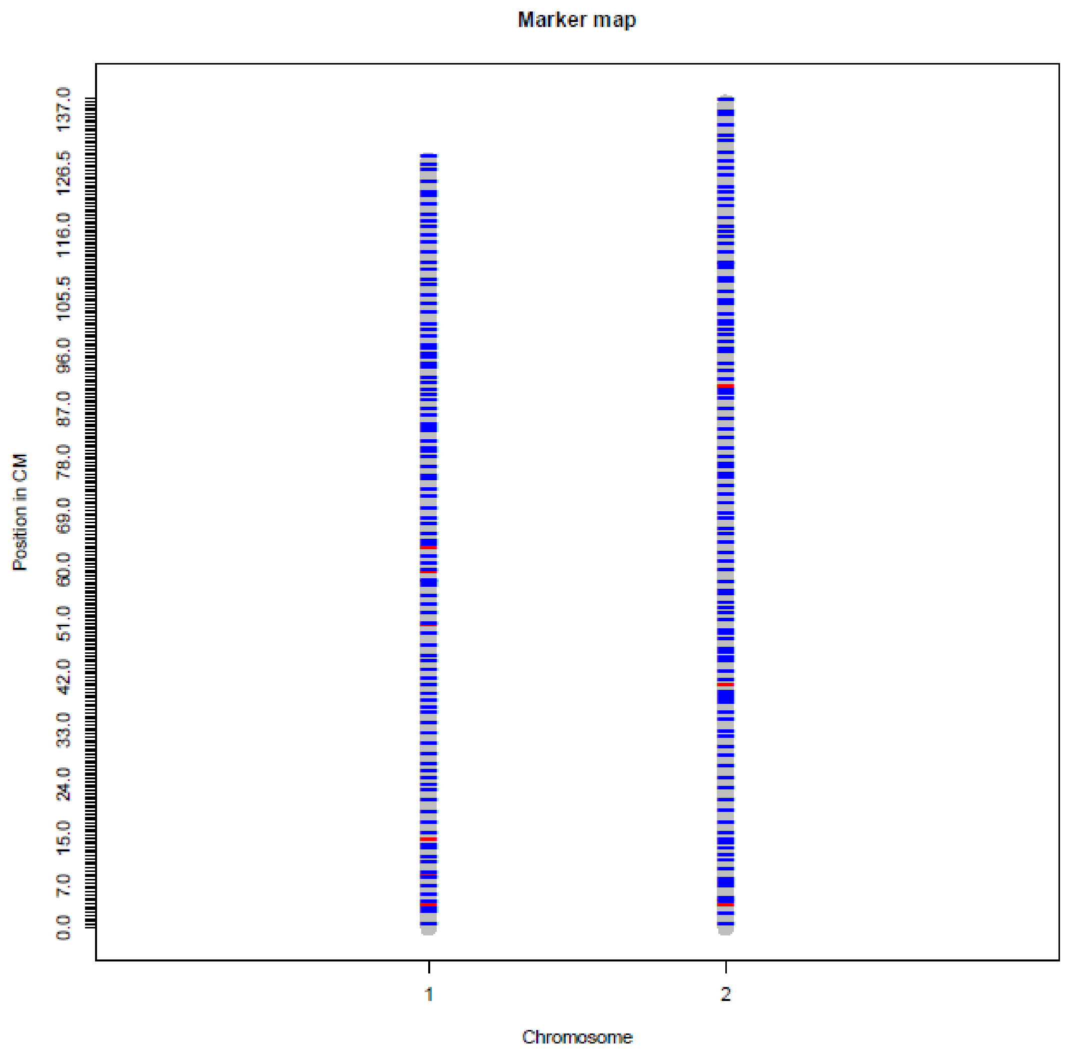


Figure 2. The simulated chromosome maps. (*Markers are indicated in blue and QTL in red. QTL were simulated at different positions but were always unlinked when more than one was used. If QTL are in the same chromosome, they kept far apart to be unlinked reasonably*).

The founder populations had four completely homozygous individuals, but they had different alleles at all loci. There were 2 males and 2 females and the sex ratio throughout simulation was kept at 1:1 in all subsequent generations. QTL were simulated at 47 cM in the first chromosome and 131 cM in the second chromosome for two loci *epistasis* model. For three loci *epistasis* model, QTL were simulated at 6cM and 47cM in the first chromosome and at 131cM in the second chromosome. The positions of simulated QTL for both two and three loci *epistasis* model were the same for every repeated simulation. Since no selection is performed, allelic frequencies remain the same as in the founder population, and allowing random mating keeps the genotype frequencies to be in equilibrium. The created maps, as shown in Figure 1(a, b, c), were simulated with noise level of 1, 3, $10^{4.5}$, 10^{5}, $10^{5.5}$ and 10^{6} for population size of 500 at generation 30 for two loci *epistasis* model. *vGWAS* – an R statistical package was used to analyze the simulated data. The simulated data contains both genotype and phenotype information. Genotype-phenotype maps are allelic combinations expressing a specific phenotypic value. The numeric values of each cell in Figure 1 are phenotypic values of their allelic combinations. The *vGWAS* program scanned the genotypes of the individuals for all possible pair wise associations. The outputs were stored and graphs were plotted. The whole process can be summarized as the following flow chart in Figure 3.

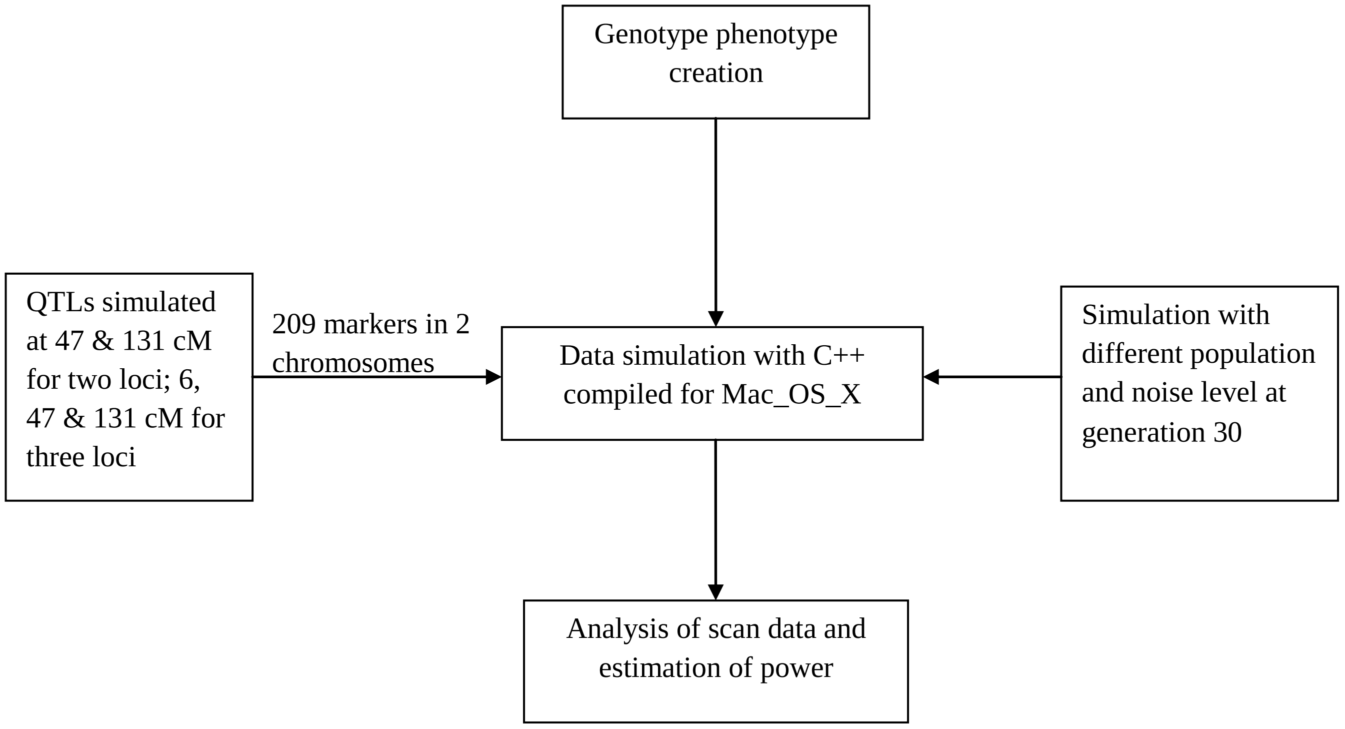

Genotype phenotype creation
QTLs simulated at 47 & 131 cM for two loci; 6, 47 & 131 cM for three loci
209 markers in 2 chromosomes
Data simulation with C++ compiled for Mac_OS_X
Simulation with different population and noise level at generation 30
Analysis of scan data and estimation of power

Figure 3. Flow chart summarizing the methodology.

A total of 1060 simulations were performed for the two loci model. Subsequently, further 120 simulations for three loci *epistasis* model were simulated (see Table 2) following the same procedure as two loci simulations (*i.e.*, for 30 generations and population sizes of 1000) for different genotype-phenotype maps. To evaluate power in terms of different heritability levels, the values of heritability and power at each of the noise levels were used.

Table 2. Simulations performed for two and three locus *epistasis* based on different population size and noise level at generation 30.

Loci	**Noise level**	**Number of Simulations**	**Population sizes**
Two loci	1	90	500
	1	140	1000
	3	90	500
	3	140	1000
	$10^{4.5}$	150	500
	10^{5}	150	500
	$10^{5.5}$	150	500
	10^{6}	150	500
Three loci	1	60	1000
	3	60	1000

2.3 Determining significant QTL:

The QTL detected by *vGWAS* program were recorded as correct or wrong QTL. Experience has shown that it is appropriate to take 6cM as a boundary between correct or wrong QTL. If the QTL detected were 6cM or less away from the position in which the QTL were simulated, it was recorded as correct QTL. Beyond that, the signals were recorded as wrong QTL. The simulated number of QTL was calculated as: number of simulations × number of interacting loci. For example, number of simulations for two loci was 90 and number of interacting loci was 2.

Therefore, the number of simulated QTL was 180. If the detected QTL were significantly above Bonferroni corrected threshold, it was recorded as a potential QTL. Significance level was calculated as α/n where α is the degree of significance (*i.e.*, 0.05) and n is the total number of markers used (*i.e.*, 209). In two loci *epistasis* model case, if the QTL are more than two or in wrong positions compared to the simulated QTL positions, then we record it as false positive QTL. The same procedure was also applied for three loci *epistasis* model. The power of *vGWAS* was calculated by using correct and wrong QTL compared to the simulated number of QTL.

2.4 Heritability and power estimation:

Traditionally, phenotypic variance is the sum of genetic and environmental variances.

$$V_P = V_G + V_E \qquad [3]$$

But, V_G may be further decomposed in to genetic variance due to mean difference (V_M) including additive (V_A) and dominance variance (V_D), and variance due to a change in variance (V_V) which includes interaction variance (V_I) and heritable variance heterogeneity (*i.e.*, variance shift between genotypes) (Shen *et al*., 2011).

$$V_G = V_M + V_V \qquad [4]$$

Broad sense heritability (H^2) can be defined as the ratio of genotypic variance (V_G) to the total phenotypic variance (V_P):

$$H^2 = V_G / V_P \qquad [5]$$

Broad sense heritability was calculated using R for different noise levels that correspond to environmental variance. Broad sense heritability includes all possible genetic effects such as *epistasis*, and additive and dominance variances. In *vGWAS*, V_I is included for calculating heritability whereas GWAS misses V_I for heritability assessment. Phenotypic and genotypic variance calculations are given in Figure 4.

```
H2 <- function(geno.pheno.map, env.var) {
        freq <- tcrossprod(c(.25, .5, .25))
        Ey <- sum(colSums(geno.pheno.map*freq))
        Ey2 <- sum(colSums((geno.pheno.map**2 + env.var)*freq))
        Vy <- Ey2 - Ey**2
        VG <- Vy - env.var
        H2 <- VG/Vy
        return(list(H2 = H2, VG = VG))
}
```

Figure 4. Heritability calculation in R

Power was estimated using the number of correct QTL out of simulated number of QTL. False positive rates (FPR) were also calculated as the number of false positive QTL recorded out of expected QTL.

Power = # detected *epistasis* / # simulated *epistasis* [6]

FPR = # false positive *epistasis*/# simulated *epistasis* [7]

2.5 Model description:

vGWAS was developed with the aim to search for variance controlling genes. Mapping variance controlling genes could be used to identify loci which have significant effect on the variance of a trait of interest and to validate such candidate *epistatic* genes. Brown-Forsythe's test of equality of variances is applied in *vGWAS*. Brown-Forsythe test of equality of variance was based on Levene's test of homogeneity of variances in k groups. It applies ANOVA statistics (Kruskal-Wallis ANOVA). The method is based on variance heterogeneity which helps to screen potentially interacting single nucleotide polymorphisms (SNPs). Brown-Forsythe (Levene's) test was used to assess the equality of group variance for variance controlling loci. For instance, for the phenotypic value Z_{ij} where i = 1….n and j = 1…m, the absolute deviation of each genotype from the median can be given as:

$$Z_{ij} = |Y_{ij} - \bar{Y}_{i.}| \qquad [8]$$

where, Y_{ij} is the value of Y for the j^{th} observation of i^{th} sub group and $\bar{Y}_{i.}$ is i^{th} sub group mean. Levene's test assumes that the null population variances are equal. The test statistics can be given as:

$$W = \frac{(N-K)\sum_{i=1}^{K} N_i(\bar{Z}_{i.} - \bar{Z}_{..})^2}{(K-1)\sum_{i=1}^{K}(Z_{ij} - \bar{Z}_{i.})^2} \qquad [9]$$

where, $\bar{Z}_{i.}$ is group mean of Z_{ij} and $\bar{Z}_{..}$ is the overall mean of Z_{ij} for N population size and K sub groups. The model also used ANOVA F test to calculate p-values. The calculated p-values are used in *vGWAS* with Bonferroni corrected significance threshold (Shen *et al.*, 2011).

The sampling distribution for our parametric model which is used to estimate the proportions explained by mean and variance is given as:

$$y_i |\ \mu M, \mu V, x_i, gM, gV \sim \mathrm{N}(\mu M + gMx_i, \exp\{\mu V + gVx_i\} \qquad [10]$$

where, M and V are mean and variance with μM and μV intercepts. x_i is a covariate denoted for SNP dosage. gM and gV are the mean and variance genetic effects. The model likelihood can be given as:

$$L = \prod_i 1/(2\pi\sigma^2) \exp\{-(y_i - \mu M - gMx_i)^2/2\sigma_i^2\} \text{ where, } \sigma_i^2 = \exp(\mu V + gVx_i) \qquad [11]$$

To estimate the parameters, $\mu M, \mu V, gM, gV$, the double generalized linear model (DGLM) is used for both mean and variance parts (Shen *et al.*, 2011). This means that linear model is fitted for the mean part and generalized linear model (GLM) is fitted for variance part. In addition, DGLM consists of one ordinary least squares (OLS) for variance part and one weighted least squares (WLS) for variance part. Both mean and variance least square parts can be combined as follows:

$$R^2 = R_M^2 + (1 - R_M^2)R_V^2 \qquad [12]$$

where, R^2 corresponds to V_G (ranges from 0 to 1), R_M^2 corresponds to V_M, and R_V^2 corresponds to V_V. The missing part in the mean model is represented by $1 - R_M^2$ and R_V^2 is the proportion reaching the missing part. Sum of squares for the mean part of the model is calculated by:

$$R_M^2 = (SS_T - SS_E) / SS_T \qquad [13]$$

Where, SS_T is the total sum of squares and SS_E is the residual sum of squares. Sum of squares of the variance part of the model is calculated by using deviances:

$$R_V^2 = (D_O - D_1) / D_O \qquad [14]$$

where, D_O is the null deviance and D_1 is the deviance from the fitted generalized linear model (*i.e.* GLM has gamma distribution). Dealing with interaction effects and vQTL, vGWAS detects *epistasis* by using Brown-Forsythe test. The sampling distribution that consists of interactions for two loci a_1 and a_2 can be given as:

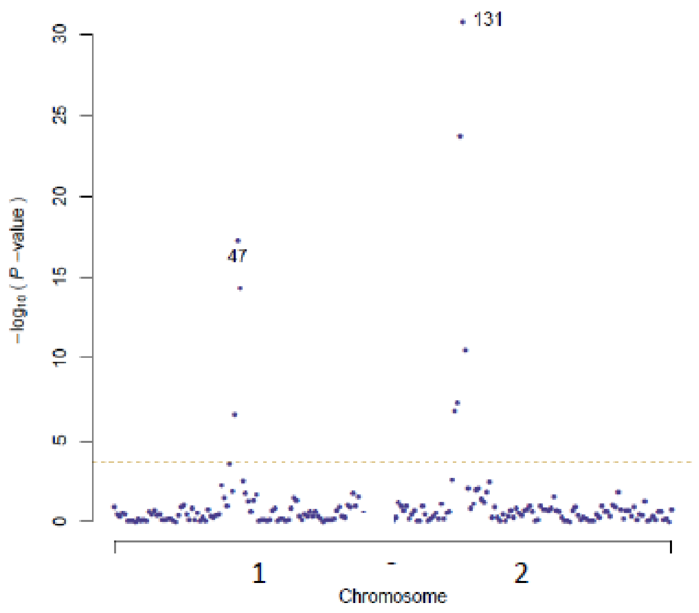
$$y_i \mid N\,(a_1x_{1i} + a_2x_{2i} + a_{12}x_{1i}x^{2i}, \sigma^2) \qquad [15]$$

where, x_{1i} and x_{2i} are covariates, a_{12} is the interaction effect and σ^2 is a common residual variance.

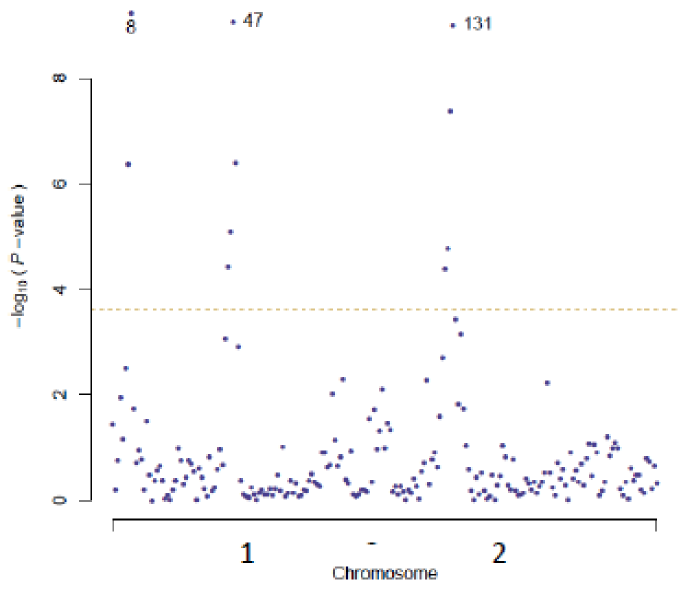
If the detected vQTL are a result of interaction effect, both G x G and G x E could be the possible factors. Testing the association of environmental factors with the phenotype confirms whether the traits are associated with the environment or not. If there is association between the traits and environmental factor, the interaction of each vQTLs and environmental factor will have significant G x E interactions. Therefore, we can also consider environmental factor as variance controlling factor. The association between the trait and environmental factor could be detected by variance heterogeneity test in our model where the noise level acts in a similar fashion as G x E interaction does (see equation 9).

3. RESULTS:

Two QTL for two loci and three QTL for three loci were observed as expected. These are shown in Figure 5a and 5b respectively. The observed positions of the QTLs were mostly the same as in the simulated data set and the same is also true for three locus *epistasis* model as shown in Figure 5. The summary of the results across different noise levels for two loci model is shown in Table 3. In this table, lower noise level results higher true positive QTL/rates and lower false positive QTL/rates whereas higher noise level results lower true positive QTL/rates and higher false positive QTL/rates. But for three loci model, only lower noise level is tested since we focused on two loci *epistasis* model primarily.



5a



5b

Figure 5. Two and three loci candidate *epistatic* QTLs. *(The light orange line is the threshold value. The x-axis represents the marker positions in the genome where number 1 and 2 refers to the first and the second chromosome.The location of the detected QTL are shown. Y-axis represents p-value as a function of base 10 logarithm with Bonferroni corrected threshold.) QTL positions were located at 47 and 131 for two loci interactions and 6, 47 and 131 cM for three locus interactions. In graph 4a and 4b noise level of 3 were used. But in 4a population size of 500 were used whereas in 4b 1000 individuals were used.)*

If the QTL were recorded beyond 100 cM, then the signal is on the second chromosome and vice versa. In the above figure, there is more than one dot (marker) above Bonferroni corrected threshold line associated with the detected QTL. This is because there are linkages between markers and they are recorded as a single peak.

Table 3. Summary of vGWAS scan for two and three loci *epistasis* model.

Loci	**No. of simulation**	**Noise levels**	**Average Heritability**	**Simulated number of QTL**	**True positive QTL**	**True positive rate**	**False positive QTL**	**False positive rate**
Two	230	1	~ 1	460	454	0.987	1	0.002
	230	3	~ 1	460	456	0.991	5	0.011
	150	$10^{4.5}$	0.84	300	268	0.893	8	0.027
	150	10^{5}	0.62	300	213	0.710	2	0.007
	150	$10^{5.5}$	0.34	300	27	0.090	4	0.013
	150	10^{6}	0.10	300	2	0.007	8	0.027
Three	60	1	-	180	163	0.906	0	0.000
	60	3	-	180	165	0.917	2	0.011

Heritability is also plotted in relation to power (see Figure 6). The graphs shown in Figure 6 are consistent with the result presented in table 3. That is at lower noise level, there are high true positive rates, power and heritability. But at higher noise level, all of these three parameters become lower. Figure 6 shows the relationship between power and heritability of genotype-phenotype maps (see Figure 1a, 1b and 1c). The trends in all three graphs are similar, but they have different turning points due to different combination of each map.

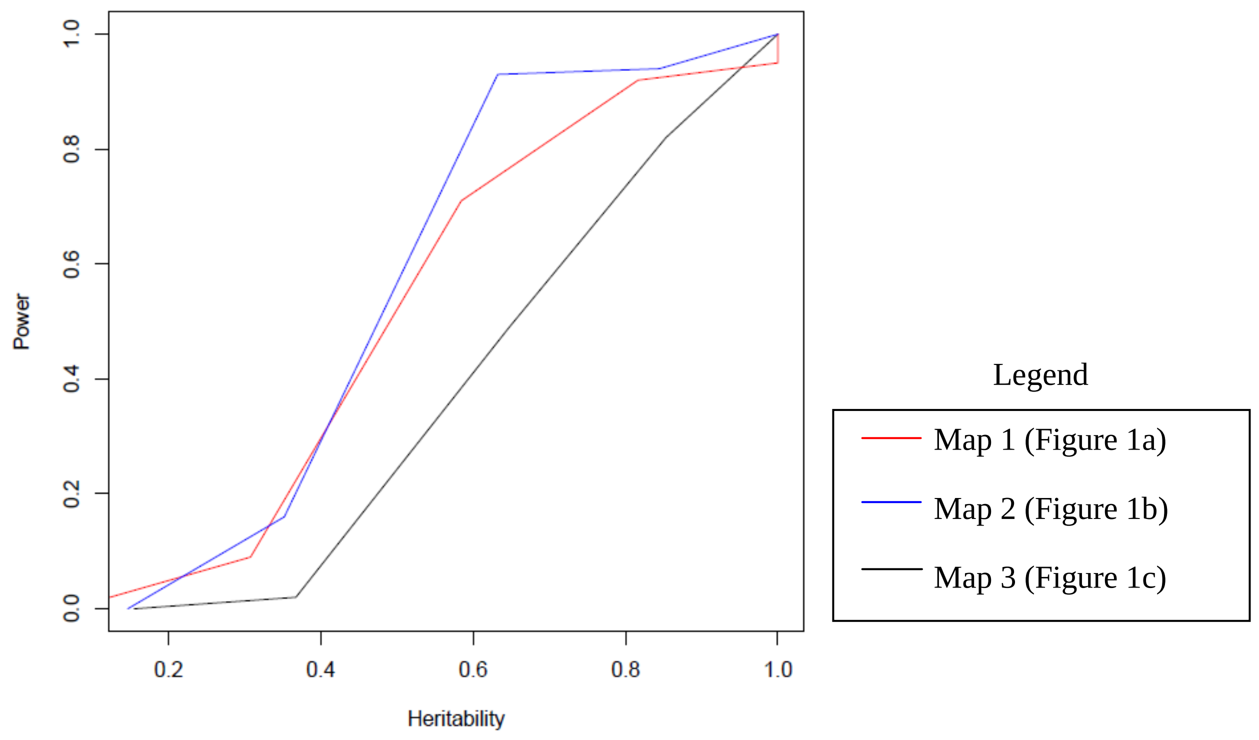


Figure 6. Power and heritability estimates for the three genotype-phenotype maps (see Figure 1 (a, b, c)).

The graph shows that the power was good at low noise level; for example, at noise level of 3, both power and heritability estimates were approximately 1.

4. DISCUSSION:

In this project, we focused on the detection of two-way interactions with a newly developed method, *vGWAS* (Shen *et al.,* 2011). We also showed that this method is able to accurately detect three loci *epistasis* interactions. The significant loci, as shown in Figure 5, could be explained as candidate epistatic loci. These loci could be detected with or without main additive effects. Current methods are unable to detect interacting QTL in the absence of main effect. The absence of main effect or detecting few QTL by the conventional association methods could be a sign of the presence of epistatic interaction. This is because detecting *epistasis* is often associated with the absence of main effect (Xu and Jia, 2007). Using the variance differences for alleles reveals an alternative mode of genetic influence with biological relevance (Pare' *et al*., 2010).

The generated data we have used for simulation follows Hardy-Weinberg equilibrium throughout the population. This would make some of the genotype to become frequent than others. Because of unequal frequencies of genotypes, genotype-phenotype maps created to show variance differences in the absence of mean difference (see Figure 1b) could also contain mean differences. This is because some of the genotypes are more frequent than the other. Although the phenomena could not greatly affect the variance effect, the incident might slightly underestimate the power analysis. In non Hardy-weinberg population, which may keep all allele frequencies equal, the effect of variance differences could be detected without mean difference by *vGWAS*. This is one potential area for improvement in the design of this project.

The increasing and decreasing trends of both power and heritability also reflects the effect of all non-genetic variances. Since increasing or decreasing the noise level would result in detection of wrong or correct significant loci, we relate the noise level as all non-genetic variances such as environmental variance (Ev), epigenetics, penetrance or sampling error. The power of *vGWAS* to detect *epistasis* was good especially for lower noise levels (higher heritability) analysis (see Figure 6). Moreover, the value of average FPR for the two loci *epistasis* model was less than 1.4% (see Table 3). This confirms that *vGWAS* can efficiently detect *epistasis.* However, to draw inference for the three loci *epistasis* model, there were not enough simulations.

Nowadays, *epistasis* becomes a common consciousness for researches since searching markers for their individual effect may not be sufficient to identify genomic regions having significant effect. Carlborg *et al*. (2006) performed a study that allows epistatic relationship among loci. In their study, only one QTL, namely Growth9, was found when testing for individual effects of loci on body weight in chicken. However, testing for *epistatic* relationship among loci enabled them to identify additional five significant genomic regions associated with growth. Several independent studies on loci with *epistatic* relationships have been found such as those influencing obesity in mice (Stylianou *et al*., 2006), odor-guide behavior in Drosophila *melanogaster* (Sambandan *et al*., 2006), the *Arabidopsis Thaliana* metabolome (Rowe *et al*., 2008), additivity and *epistasis* controls of growth and yield in tomatoes (Causse *et al*., 2007), genetic architecture of co-variation in skull trait complexes in mice (Wolf *et al.,* 2005), genome-wide expression in yeast (Store *et al*., 2005), and QTL mapping of yeast (Nogami *et al*., 2007). Designing and monitoring novel breeding strategies that take into account interacting loci could be appropriate to increase the power of current selection schemes. Since *vGWAS* detects more QTL than GWAS does, more QTL can be used for selection in animal breeding.

When the phenotypes in the population are characterized by non-additive variance of major genes, the candidates may not be evaluated directly for the traits that are under selection. Therefore, there is no or low pressure of conventional selection on the effect of major genes segregating in the population (Dodds *et al*., 2007). Dagnachew *et al*., 2011 reported a deletion allele in exon 12 of CSN1S1 gene (*i.e.*, associated with lowered fat and protein composition in milk) which exhibits a non-additive effect. They have reported over-dominance in kg milk and lactose percentage. It has unusually high frequency in the Norwegian goat population. In their study, they explained that the phenomenon decreases the selection pressure for the allele when conventional breeding methods are used. To decrease the frequency of this deletion at national level, molecular information on the deletion is included as a selection criterion in the national breeding scheme. However, to explain the occurrence of high frequency of the allele, while the aim of breeding is against the effect of this allele (*i.e.*, deletion allele, CSN1S1), the presence of *epistasis* and/or linkage could be considered (Cesurer *et al*., 2002). This phenomenon can be an example of the gap that earlier methods had in animal breeding. Since *vGWAS* detects more QTL associated with the trait (Shen *et al.,* 2011), it would provide more information for animal selection. This means that the loci that interact epistatically for a given trait can be detected by *vGWAS*. It might be necessary to build extended segment to capture those QTL underlying the traits of interest. Since larger segments recombine more frequently, the epistatic effects of linked genes would go on the cost of stability of haplotype effect across generation. Therefore, using *epistasis* of linked genes could speed up short term selection response. In addition, the loci carrying alternative genotypes may not show mean differences. But these loci could have variance differences between alternative genotype classes and this can be detected by *vGWAS*. Using *vGWAS* could therefore fill such gaps which were not reached by earlier methods.

Usually, selection generates linkage disequilibrium. This means that for the selection of the optimum values of a given trait, certain genes will be responsible. These genes interact epistatically to give the optimum values of a trait (Phillips, 2008). Through selection, linkage and *epistasis* have evolutionary consequences over generations. Therefore, designing selection strategies based on conventional breeding scheme should consider the presence of *epistasis* and linkage.

Currently, lack of precise breeding value of any single individual is associated with addressing the source of genetic variation accurately. This means that in our breeding value estimation, we must consider whether additive combinations of a single locus could be used to explain phenotypic variation associated with multi-locus genotypes or whether important non-linear interactions exist. We also have to consider the independence of inheritance and distribution of one locus to those of the other loci. Since the expressions of polygenic traits are influenced by environmental variance, we should also consider whether gene expression varies with context and whether specific genotypes are associated with particular environment (Lynch and Walsh, 1998). Existing breeding knowledge relates response to selection closely with the level of additive genetic variance. However, environmental variance reduces the efficiency of selection process by obscuring genotype-phenotype relationship (Lynch and Walsh, 1998). Usually, low heritability is associated with high environmental effect. But, it could also be associated with traits regulated by variance controlling genes rather than mean controlling genes. Implementing *vGWAS* therefore could detect the additive effects at two loci a_1 and a_2 of variance controlling loci and the interaction effects a_{12} (see equation 15).

Detecting loci (vQTLs) which control such variance provide valuable information for various multi-disciplinary studies. For instance, in genomic selection, it can provide more consistent livestock selection schemes (Rönnegård and Valdar, 2011). This means that at the beginning of selection, a change in mean results in economic gain. When the optimum is reached through time, selection pressure may shift from the mean to variance. Reducing variance could further promote economic gain through uniformity (Mulder *et al*., 2008). Animal uniformity for traits with nearly optimum values have economic interest such as pH range, litter size, weight and quality of carcass in pigs, sheep and broilers (Mulder *et al*., 2008). Moreover, *vGWAS* also helps to find those genetic factors underlying the disease phenotypes since most complex diseases are caused by combined effect of multiple genes.

As a shortcoming of this method, *vGWAS* requires more observations (*i.e.,* fivefold as many observations as mean controlling loci) to reach the same precision (Rönnegård and Valdar, 2011). Since *vGWAS* is a conservative approach, it has low power to detect interactions at lower heritabilities. This means that *vGWAS* requires higher heritability as shown in Figure 6 to get a powerful estimate compared with other methods. But, most quantitative traits have low to moderate heritability estimates which may make it difficult to detect interactions with good power.

5. CONCLUSION

Our results show that *vGWAS* is a powerful and promising tool for candidate *epistatic* gene detection. It would be applicable for real data dealing with complex disease or obscured genetic interactions. However, the requirement of high heritable traits could face a challenge in its applicability for most of quantitative traits. The identified candidate *epistatic* genes would require further experimental procedures to confirm that the candidates are the actual *epistatic* genes. The missing heritability due to variance shift between genotypes and interaction variance linked with *epistasis* could be reachable by *vGWAS* since the method has good power and accuracy for highly heritable traits. Although we have also tested three loci *epistasis* model, the performed simulations were not enough due to its complexity. Therefore, more simulation is required to draw the power analysis for three loci *epistasis* model. Practically, it is possible to reanalyze previous datasets by *vGWAS* to detect significant loci which were not detected by GWAS.

6. REFERENCES

Alvarez-Castro, J. M. and Carlborg, O., 2007. A unified model for functional and statistical *epistasis* and its application in quantitative trait loci analysis. *Genetics* **176**: 1151–1167.

Carlborg, Ö., Andersson, L. and Kinghorn, B., 2000. The use of a genetic algorithm for simultaneous mapping of multiple interacting quantitative trait loci. *Genetics* **155**: 2003–2010.

Carlborg, Ö., Jacobsson, L., Ahgren, P., Siegel, P. and Andersson, L., 2006. *Epistasis* and the release of genetic variation during long-term selection. *Nature Genet*. **38**: 418-420.

Carlborg, Ö. and Haley C. S., 2004. *Epistasis*: too often neglected in complex trait studies? *Nature Rev. Genet*. **5**: 615- 625.

Carlborg Ö., Kerje, S., Schütz, K., Jacobsson, L., Jensen, P. and Andersson, L., 2003. A global search reveals *epistatic* interaction between QTLs for early growth in the chicken. *Genome Res*. **13**: 413–421.

Causse, M., Chaïb, J., Lecomte, L., Buret, M. and Hospital, F., 2007. Both additivity and *epistasis* control the genetic variation for fruit quality traits in tomato. *Theor. Appl. Genet.* **115**: 429–442.

Cesurer, L., Bölek, Y., Dokuyucu, T., Akkay, A., 2002. Understanding of heterosis. *KSUJ. Science and Engineering* **5**(2): 68-75.

Chase, K., Adler, F. R. and Lark, K. G., 1997. Epistat: a computer program for identifying and testing interactions between pairs of quantitative trait loci. *Theor. Appl. Genet*. **94**:724–730.

Cordell, H. J., 2009. *Epistasis*: what it means, what it doesn’t mean, and statistical methods to detect it in humans. *Human Molecular Genetics* **11**: 2463-2468.

Cordell, H. J., Todda, J. A., Hilla, N. J., Lorda, C. J., Paul, A. Lyonsa, P. A., Peterson, L. B., Wickerb, L. S. and Claytona, D. G., 2001. Statistical modeling of inter-locus interactions in a complex disease: rejection of the multiplicative model of *epistasis* in type 1 diabetes. *Genetics* **158**: 357–367.

Corvin, A., Craddock, N., Sullivan, P. F., 2010. Human genome-wide association studies a primer. *Psychol. Med.* **40**(7): 1063-77.

Crow, J. F., 2010. On *epistasis*: why it is important in polygenic directional selection. *Phil. Trans. R. Soc.* **365**: 1241-1244.

Crow, J. F. and Kimura, M. 2009. An Introduction to population genetics theory. Caldwell, NJ: Blackburn Press.

Dagnachew, B. S., Thaller, G., Lien S., and Ådnøy, T., 2011. Casein SNP in Norwegian goats: additive and dominance effects on milk composition and quality. *Genetics Selection Evolution* **43**:31 doi:10.1186/1297-9686.

Deng, H.W., Chen, W.M. Recker, R.R., 2000. QTL fine mapping by measuring and testing for hardy-weinberg and linkage disequilibrium at a series of linked marker loci in extreme samples of populations. *Am. J. Hum. Genet.* **66**:1027–1045.

Dodds, K. G., McEwan, J. C., and Davis G. H., 2007. Integration of molecular and quantitative information in sheep and goat industry breeding programmes. *Small Ruminant Res*. **70**:32-41.

Evans, D. M., Marchini J., Morris A. P., Cardon L. R., 2006. Two-stage two-locus models in genome-wide association. *PLoS Genet.* **2**(9): e157.

Fisher, R. A., 1918. The correlation between relatives on the supposition of mendelian inheritance. *Trans. R. Soc. Edinb*. **52**: 399-433.

Jiang L, Liu J, Sun D, Ma P, Ding X, et al. (2010) Genome Wide Association Studies for Milk Production Traits in Chinese Holstein Population. *PLoSONE* 5(10): e13661. doi:10.1371/journal.pone.0013661.

Hoh, J., Wille, A., Zee, R., Cheng, S., Reynolds, R., Lindpaintner, K., and Ott, J., 2000. Selecting SNPs in two-stage analysis of disease association data: A model-free approach. *Ann. Hum. Genet.* **64**(5):413–417.

Iles, M.M., 2008. What can Genome-Wide Association Studies tell us about the genetics of common disease? *PLoS Genetics* **4**:2|e33.

Ljungberg, K., Carlborg, Ö., Holmgren, S., 2004. Simultaneous search for multiple QTL using the global optimization algorithm DIRECT. *Bioinformatics* **20** (12): 1887–1895.

Lynch, M. and Walash, B., 1998. Genetics and Analysis of Quantitative Traits. Sanderland: Sinauer associates, Inc. publishers.

Moore, J. H., 2003. The ubiquitous nature of *Epistasis* in determining susceptibility to common human diseases. *Hum Hered*. **56**:73–82.

Moore, J. H. and Williams, S. M., 2005. Traversing the conceptual divide between biological and statistical *epistasis*: systems biology and a more modern synthesis. *Bioessays* **27**: 637–646.

Maher, B., 2008. The case of the missing heritability. *Nature* **456**: 18-21.

Malmberg, R., Mauricio, R., 2005. QTL-based evidence for the role of *epistasis* in evolution. *Genet. Res. Camb.* **86**: 89–95.

Manolio, T. A., Collins, F. S., Cox, N. J., Goldstein, D. B., Hindorff, L. A., Hunter, D. J., McCarthy, M. I., Ramos, E. M., Cardon, L. R., Chakravarti, A., Cho, J. H., Guttmacher, A. E., Kong, A., Kruglyak, L., Mardis, E., Rotimi, C. N., Slatkin, M., Valle, D.,Whittemore, A. S., Boehnke, M., Clark, A. G., Eichler, E. E., Gibson, G., Haines, J. L., Mackay, T. F., McCarroll, S. A., Visscher, P. M, 2009: Finding the missing heritability of complex diseases. *Nature*: **461** (7265):747–753.

Mulder, H. A., Bijma, P., and Hill, W. G., 2008. Selection for uniformity in livestock by exploiting genetic heterogeneity of residual variance. *Genet. Sel. Evol*. **40**: 37-59.

Nelson, M. R, Kardia, S. L, Ferrell, R. E, Sing, C. F, 2001: A combinatorial partitioning method to identify multilocus genotypic partitions that predict quantitative trait variation. *Genome Res*. **11**(3):458-470.

Nogami, S., Ohya, Y. and Yvert, G., 2007. Genetic complexity and quantitative trait loci mapping of yeast morphological traits. *PLoS Genet*. **3**: e31.

Omholt, S. W., Plahte, E., Oyehaug, L. and Xiang, K., 2000. Gene regulatory networks generating the phenomena of additivity, dominance and *epistasis*. *Genetics* **155**: 969–980.

Pare´ G., Cook, N. R., Ridker, P. M., Chasman, D. I., 2010. On the use of variance per genotype as a tool to identify quantitative trait interaction effects: A report from the women's genome health study. *PLoS Genet.* **6** (6). e1000981.

Phillips, P. C., 2008. *Epistasis* - the essential role of gene interaction in the structure and evolution of genetic systems. *Nature (Genetics)* **9**: 855-867.

Ritchie, M. D., Hahn, L. W., Roodi, N., Bailey, L. R., Dupont, W. D., Parl, F. F., and Moore, J. H., 2001. Multifactor-dimensionality reduction reveals high-order interactions among estrogen-metabolism genes in sporadic breast cancer. *Am. J. Hum. Genet.* **69** (1):138-147.

Wu, R., Ma, C., and Casella, G. 2007. Statistical genetics of quantitative traits: New York. Linkage, maps, and QTL. Spring science and Business media, LLC publisher.

Rönnegård, L. and Valdar W., 2011. Detecting Major Genetic Loci Controlling Phenotypic Variability in Experimental Crosses. *Genetics*. **188** (2): 435-447.

Rowe, H. C., Hansen, B. G., Halkier, B. A. and Kliebenstein, D. J., 2008. Biochemical networks and *epistasis* shape the *Arabidopsis thaliana* metabolome. *Plant Cell* **20**: 1199–1216.

Sambandan, D., Yamamoto, A., Fanara, J. J., Mackay, T. F. and Anholt, R. R., 2006. Dynamic genetic interactions determine odor-guided behavior in Drosophila melanogaster. *Genetics* **174**: 1349-1363.

Shen, X., Petterson, M., Rönnegård, L. and Carlborg, Ö., 2011. Inheritance beyond plain heritability: Variance-controlling genes in *Arabidopsis Thaliana.* Submitted.

Storey, J. D, Akey. J. M, Kruglyak. L., 2005. Multiple locus linkage analysis of genome-wide expression in yeast. *PLoS Biol.* **3**: e267.

Stylianou, I. M., Korstanje, R., Li, R., Sheehan, S., Paigen, B., Churchill, G. A., 2006. Quantitative trait locus analysis for obesity reveals multiple networks of interacting loci. *Mamm. Genome* **17**: 22-36.

Wade, M. J., Winther, R. G., Agrawal, A. F. and Goodnight, C. J. (2001). Alternative definitions of *epistasis*: dependence and interaction. *TRENDS in Ecology & Evolution* **16** (9): 498-504.

Wolf, J. B., Leamy, L. J., Routman, E. J. and Cheverud, J. M., 2005. *Epistatic* pleiotropy and the genetic architecture of co-variation within early and late-developing skull trait complexes in mice. *Genetics* **171**: 683–694.

Xu, S. and Jia, Z., 2007, Genome-wide analysis of *epistatic* effect for quantitative traits in barley. *Genetics* **175**: 1955-1963.

Yang, Q., Khoury, M. J., Sun, F. and Flanders, W. D., 1999. Case-only design to measure gene -gene interaction. *Epidemiology* **10**: 167–170.

Zhang, X., Pan, F., Xie, Y., Zou, F. and Wang, W., 2010. COE: A general approach for efficient genome-wide two-locus *epistasis* test in disease association study. *Journal of Computational Biology* **17**(3): 401–441.

ACKNOWLEDGEMENT

First, let me take a moment of praise to the Almighty God for sending miracles every single day in my life. I thank Holy Virgin Mary, the Mother of Jesus Christ for being a shed all the way here.

This thesis would not have looked the way it is right now had it not been for my great supervisors. I would like to thank Dr. Ronnie Nelson, my main supervisor, for his valuable encouragement and motivation under his supervision. I am truly grateful for your critical advices and for keeping me in track in writing this thesis. My gratitude also passes to my co-supervisors; Xia Shen for your support during the analysis in R and for statistical aspects of the project and Tormod Ådnøy, Associate Professor in department of Animal and Aquaculture Science (UMB) in Norway, for your constructive advices in the writing process and for the general progress of my paper. I am also indebted to forward my gratitude for Örjan Carlborg, my examiner, for your counseling. Last but not least, I would like to thank all computational genetics group in general for their support, advice and for making my time during my thesis work enjoyable.

APPENDIX

C++ simulation program

```
 ronnie.nelson@hgen.slu.se
 This program simulates the creation of offspring
 from set or user defined genotypes (haplotypes) for analysis of
 qtl estimation programs */

/* Note that all the arrays (elements in the arrays) start at one with zero as an empty space to use in some
 cases for sorting or finding things or resetting some values before selection*/

#include <iostream>
#include <string>
#include <algorithm>
#include <math.h>

using namespace std;

//structure
struct saved_information {//should have the info for all the subpops, to get the freq just devide by
subpopsize or 2*subpopsize for allele freq or the number of loci
	int prop_fixed_loci; //the number of fixed alleles in this population
	int *one_allele_freq; /*determine the allele freq of the 1st allele,
								can use to determine if it is fixed and if the markers
								are snps the other freq can also be detemined*/
};

struct chromosome {
	char chr_name[40]; //name of the chromosomes
	int markernumber; //including the QTL's
	int qtl_per_chr; //the number of qtls in this chr
	string *markernames; //array of marker names
	float *maleCM; //array of positions of markers cM (first always 0) this is the absolute position on
the chr
	float *femaleCM; //seperate for male and females??
	float *RF_to_next_marker_maleCM;//the recombination frequency between markers next to e.o.,
will enable some different ways to calc
	float *RF_to_next_marker_femaleCM;
	bool *type; //marker or qtl, marker = 1, qtl = 0, may change to int when microsats are used
	  //add a fixed for an allele function..., cpuld be an array here?
	  //sex chromosome
     //int *physical position??
};

struct qtl_array {
	char qtl_name[40]; //name of the phenotype
	int qtl_chro; //which chromosome
	float qtl_male; //the place on the male map
	float qtl_female;
	int qtl_pos;//number on the map of where qtl is locates i.e. marker number 200 is a qtl
	/*
	 add here the amount of additive and or dominance varianve is explaind by the qtl
	 also which if there are more than one trait which trait it affects
	 */
};

struct individual{//all should will be part of a population
	bool sex; //1 defines male, 0 defines female
	int subpopID;//could maybe remove this to make faster!!
	int *haplotypeA, *haplotypeB; // the actual genotype composition of the individual
	int times_mated; //
	bool selected_to_mate; //if this is 1 the individual can mate if 0 not
	int father, mother;
	float phenotype, phenotype2;
	//string ind_id;//no name yet for individuals
};

struct population{
	char popname[40];
	int popsize;
	individual *ind;
};

//GLOBAL VARIABLES
//(From the Inputfile)
int chromosome_number;
int qtl_number;
int total_markernumber; //excluding qtls
int haplotype_size; //markers and qtls
int burnin_generations;
string marker_type; //snp, micro_sat
chromosome *chr;
qtl_array *qtl;

population founderpop;// the founder population
population *subpop;//the array of subpops creating the parental line
population *subpop_offspring;
int population_size_total;//total no of individuals in whole population
int subpopnumber;//the number of subpops during burnin
int subpopsize;// all subpopsizes are currently assumed to be the same, make this an array if different
popsizes are to be simulated
char rec_freq_method[2];//the way the recombination frequency is calculated (h haldane, k kosambi, e
map distance = recmbination frequency), currently only haldane

//saved info
saved_information **info_per_gen_per_pop;//2dimentional 1st[generation] 2nd[subpopulation]
int *tot_fixed_loci;//array per gen
int *same_fixed_loci;//array per gen
int *sexratio;//array per gen, actually the number of males

//noise for the phenotypes
//int *unifdistr; // array of uniforma distr numbers
float *int_noise;
float noise_scale; //used to scale the error, currently to around one stdev

//Function: draw the chromosome maps
void plot_fixed_allel_freq(){
	//var
	int counter, counterB, counterC;
	FILE * outfile;

	//creating a R-info file named: "plot_freq2.R"
	outfile = fopen("plot_freq2.R","w");
	fprintf(outfile, "#data to be used in R-file plot_freq.R");
	fprintf(outfile, "\npopsize <- %d",population_size_total);
	fprintf(outfile, "\nsubpopnumber <- %d",subpopnumber);
	fprintf(outfile, "\nloci <- %d",haplotype_size);
	fprintf(outfile, "\nmaxgen <- %d",burnin_generations);
	fclose(outfile);

	//creating the data file named: "outfreq.grph"
	outfile = fopen("outfreq.grph","w");
	fprintf(outfile, "generation\tmales\ttotal_fixed\tsame_allele\tdifferent_allele");
	for (counter = 1; counter<(subpopnumber+1); counter++) {
		fprintf(outfile, "\tfixed_pop_%d",counter);
	}

	//adding all the allele freq of all the populations to the recorded data
	for (counter = 1; counter<(subpopnumber+1); counter++){
		for (counterB = 1; counterB<(haplotype_size+1); counterB++) {
			fprintf(outfile, "\tpop_%d",counter);
			fprintf(outfile, "locus_%d",counterB);
		}	}

	for (counter =1; counter<(burnin_generations+1); counter++) {//for each gen recorded
		fprintf(outfile,
"\n%d\t%d\t%d\t%d\t%d",counter,sexratio[counter],tot_fixed_loci[counter],same_fixed_loci[counter],(tot
_fixed_loci[counter]-same_fixed_loci[counter]));
		for (counterB = 1; counterB<(subpopnumber+1); counterB++) {//for each subpop
			fprintf(outfile,
"\t%d",info_per_gen_per_pop[counter][counterB].prop_fixed_loci);
		}

		//adding all the allele freq of all the populations to the recorded data
		for (counterB = 1; counterB<(subpopnumber+1); counterB++){
			for (counterC = 1; counterC<(haplotype_size+1); counterC++) {
```

```
                                        fprintf(outfile,
"\t%d",info_per_gen_per_pop[counter][counterB].one_allele_freq[counterC]);
                                }
                        }

                }

                fclose(outfile);

                system ("R CMD BATCH plot_freq.R R_log_freq_plot.txt");
//              system("open proportion_loci_fixed.pdf");
}

//Function: draw the chromosome maps
void plot_marker_info(){
                //var
                int counter, counterB, counterC;
                FILE * outfile;

                //creating a outfile, should go in a seperate function when more than one plot is created
                outfile = fopen("outin.grph","w");
                fprintf(outfile, "markerno\tchr\tposition\ttype");
                counterC = 1;

                for (counter = 1; counter<(chromosome_number+1); counter++) {
                        for (counterB = 1; counterB<(chr[counter].markernumber+1); counterB++) {

                                fprintf(outfile,"\n",counterC);
                                if (chr[counter].type[counterB] == 0) {

                fprintf(outfile,"%d\t%d\t%f\tq",counterC,counter,chr[counter].maleCM[counterB]);
                                }
                                else {

                fprintf(outfile,"%d\t%d\t%f\tm",counterC,counter,chr[counter].maleCM[counterB]);
                                }
                                counterC =counterC+1;
                        }
}
                fclose(outfile);
                system ("R CMD BATCH draw_qtls.R R_log_chr_draw.txt");
//              system("open chromosome_map.pdf");
}

//Function: show the imported data: can use to create a log file, alternative output
void show_marker_info(){
                //var
                int counter, counterB;
                //cout << "\nThe data imported (also written to the following log file '*.txt'):\n";
```

```
	//QTL OUTPUT
	cout << "\n";
	cout << "Number of QTLs: " << qtl_number << "\n";

	for (counter=1; counter<(qtl_number+1); counter++) {
		cout << qtl[counter].qtl_name;
		cout << ", on chromosome " << qtl[counter].qtl_chro << ", at positions "
			<< qtl[counter].qtl_male << " and " << qtl[counter].qtl_female << "
(male,female)\n";
	}

	//CHROMOSOME OUTPUT, including the places of the QTLS
	cout << "\n";
	cout << "Number of markers (total, excluding QTL's): " << total_markernumber  << " \n";
	cout << "Number of markers including QTL's: " << (haplotype_size) << " \n";
	cout << "Marker type: " << marker_type << "\n";
	cout << "Number of chromosomes: " << chromosome_number << "\n";

	for (counter=1; counter<(chromosome_number+1); counter++) {
		cout << chr[counter].chr_name <<"\n";
		cout << "Marker no (incl. QTL): " << chr[counter].markernumber <<"\n";
		cout << "QTL no: " << chr[counter].qtl_per_chr <<"\n";

		//slightly inefficient to repeat a few times but
		//this is mainly for tessting and
		//may want to keep in this way to make the rows into cols if wanted
		cout << "Marker_names:";
		for (counterB=1; counterB<(chr[counter].markernumber+1); counterB++) {
			cout << " " <<chr[counter].markernames[counterB] ;
		}
		cout << "\n";
		cout << "Male_map_(cM):";
		for (counterB=1; counterB<(chr[counter].markernumber+1); counterB++) {
			cout << " " <<chr[counter].maleCM[counterB] ;
		}
		cout << "\n";
		cout << "Female_map_(cM):";
		for (counterB=1; counterB<(chr[counter].markernumber+1); counterB++) {
			cout << " " <<chr[counter].femaleCM[counterB] ;
		}
		cout << "\n";
		cout << "Recombination_frequency_to_next_marker_(male,cM):";
		for (counterB=1; counterB<(chr[counter].markernumber+1); counterB++) {
			cout << " " <<chr[counter].RF_to_next_marker_maleCM[counterB] ;
			if (chr[counter].RF_to_next_marker_maleCM[counterB] < 0) {
				cout << "\n\n****ERROR: check map as markers are not in positional
order\n\n";
			}
		}
		cout << "\n";
		cout << "Recombination_frequency_to_next_marker_(female,cM):";
		for (counterB=1; counterB<(chr[counter].markernumber+1); counterB++) {
			cout << " " <<chr[counter].RF_to_next_marker_femaleCM[counterB] ;
			if (chr[counter].RF_to_next_marker_femaleCM[counterB] < 0) {
				cout << "\n\n****ERROR: check map as markers are not in positional
order\n\n";
			}
		}
		cout << "\n";
		cout << "Marker/QTL:";
		for (counterB=1; counterB<(chr[counter].markernumber+1); counterB++) {
			if (chr[counter].type[counterB] == 0) {
				cout << " Q";
				}
			else {
				cout << " M";
			}
		}
		cout << "\n";
	}
	cout << "\n";
}

//Function: show the individuals in a population
void showpop(population which_pop){//passing a population here to show what is in it
	//var
	int counter, counterB;
	float sexratio_o;
	string sexer, outstring1;
	//char inttochar[40];

	sexratio_o = 0;

	cout << "\nInformation for population: "<< which_pop.popname <<"\n";
	cout << "Number of inidivuduals in population: " << which_pop.popsize <<"\n";
	cout << "\nIndividual info:\n";

	outstring1 = "Ind PopID Sex Markers";
	/*for (counter =1 ;counter<(chromosome_number+1); counter++) {
		for (counterB =1 ;counterB<(chr[counter].markernumber+1); counterB++) {
			sprintf(inttochar,"%d",counter);
			outstring1 = outstring1 + " (c" + inttochar + ")_" +
chr[counter].markernames[counterB];
		}
	}*/
	cout << outstring1 << "\n";

	for (counter = 1; counter<(which_pop.popsize+1);counter++) {

		if (which_pop.ind[counter].sex == 1) {
				sexer =" MALE ";
				sexratio_o = sexratio_o + 1;
			} else {
				sexer =" FEMALE ";
			}
			cout << "Ind_"<<counter<< sexer<< which_pop.ind[counter].subpopID << ", can mate = 
" << which_pop.ind[counter].selected_to_mate <<", times_mated "<< 
which_pop.ind[counter].times_mated << " ";
			 //---SHOWING THE GENOTYOPE
			for (counterB = 1; counterB<(haplotype_size+1);counterB++) {
				cout << which_pop.ind[counter].haplotypeA[counterB] << "/" 
<<which_pop.ind[counter].haplotypeB[counterB] << " ";
			}
			cout << " \n";
		}
		//can be made to show only part of the output for a given individual
		sexratio_o = sexratio_o/(which_pop.popsize);
		cout << "Population sex ratio: " << sexratio_o;
		cout << "\n";
}

//Function: show the files in the current directory
string showfiles(){
		//may want to change this that the user can choose the file from any dir
		//var
		char openfile[256];
		cout<< "\nChoose the main information file to open from the current directory:\n";
		system("ls");
		cout << "\nType the complete name and extention and press enter, ";
		cout << "type 'Q' to quit: ";
		cin.getline(openfile,256);
		return(openfile);
}

//Function: splice the qtl and marker array togehter
void spliceqtl (){
		//var
		int counter, counterB, counterC, counterD;
		chromosome tempchr, tempchrB;
		qtl_array *CQarr; // and array of qtls within each chr, as the no of qls are only few this should 
never be very big and this can be created each time
		int marker, incerA, incerB;
		float map_dist_male, map_dist_female; //to calculate the recombination frequency
		string warning_out = "";

		//INSERTING THE QTLS ON THE MAP (male map as reff first), This is tricky as the two 
arrays not clearly defined needs to be merged
		/* Note that the male map is used an not both the male and female map
		 this means the assumprion is made that the qtls are in similar order on
		 both the male and female maps. If this is not the case, a few things sould be
```

```
 done in this application to rectify. But shoud not be a problem*/

incerA = 1;
incerB = 1;

for (counter=1; counter<(chromosome_number+1);counter++) {//cycle through chromosomes

        CQarr = new qtl_array[qtl_number+1];//again inefficient but I expect not a lot of qtls
        marker = 1;
        for (counterB = 1; counterB<(qtl_number+1); counterB++) {//cycle through qtls
                if (qtl[counterB].qtl_chro == counter) {
                        CQarr[marker] = qtl[counterB];
                        marker++;
                }
        }

        for (counterB=marker; counterB<(qtl_number+1); counterB++) {
                CQarr[counterB].qtl_male = -2;//this is used in the empty array REMEBER -2,
the empty values
        }

        //SORT the sub qtl arrays here if needed, and not in propper order when on input
SORT HERE

        /* checking for (counterB = 1; counterB<(qtl_number+1); counterB++){//checking
function
                cout << "\nchromosome " << counter << " " << CQarr[counterB].qtl_name << "
"<< CQarr[counterB].qtl_male;
        }*/

        chr[counter].type = new bool[chr[counter].markernumber+1];//to get the marker type
        tempchr = chr[counter];//for each chr a temp holder is made with all the info of the chr
        tempchrB.markernames = new string[chr[counter].markernumber+1];
        tempchrB.maleCM = new float[chr[counter].markernumber+1];
        tempchrB.femaleCM = new float[chr[counter].markernumber+1];
        tempchrB.type = new bool[chr[counter].markernumber+1];
        /*hierdie hierbo is kak, veral die tempchrB en het 3 dae gevat om uit te figure maar...
        it is needed as the tempchr and chr[counter] becomes
        exactly equal interchange values in a way I can't predict*/


        /* cheking for (counterB=1; counterB<(tempchr.markernumber+1);counterB++)  {
                cout << "\nTempchr marker number " << counterB << " name: " <<
tempchr.markernames[counterB];
        }*/

        chr[counter].type = new bool[chr[counter].markernumber+1];//to get the marker type

        counterC = 1;
        counterD = 1;
```

```
			for (counterB = 1; counterB<(chr[counter].markernumber+1); counterB++) {
				if ((CQarr[counterC].qtl_male <= tempchr.maleCM[counterD]) and 
(CQarr[counterC].qtl_male > 0)){
					tempchrB.markernames[counterB] = CQarr[counterC].qtl_name;
					tempchrB.maleCM[counterB] = CQarr[counterC].qtl_male;
					tempchrB.femaleCM[counterB] = CQarr[counterC].qtl_female;
					tempchrB.type[counterB] = 0;
					qtl[incerB].qtl_pos = incerA;
				//	cout << "\n MARKER(qtl): " << CQarr[counterC].qtl_name <<  " 
counterB: " << counterB <<  " counterC: " << counterC << " counterD: " << counterD;
					counterC = counterC+1;
					incerB = incerB+1;
				}
				else {
					tempchrB.markernames[counterB] = tempchr.markernames[counterD];
					tempchrB.maleCM[counterB] = tempchr.maleCM[counterD];
					tempchrB.femaleCM[counterB] = tempchr.femaleCM[counterD];
					tempchrB.type[counterB] = 1;
				//	cout << "\n MARKER(marker): " << tempchr.markernames[counterD] 
<<  " counterB: " << counterB <<  " counterC: " << counterC << " counterD: " << counterD;
					counterD = counterD+1;
				}
				incerA = incerA+1;
			}

			chr[counter].markernames = tempchrB.markernames; //replacing the values in the used 
array nl. chr[counter]
			chr[counter].maleCM = tempchrB.maleCM;
			chr[counter].femaleCM = tempchrB.femaleCM;
			chr[counter].type = tempchrB.type;

			//the relative positions
			for (counterB = 1; counterB<(chr[counter].markernumber+0); counterB++) {/*note this 
+0, because we are working with the 

							intervals and this is one less than the 

							number of markers*/
				/*here we calculate the recombination frequency dependant on the map distance
				 three methods can be used, chosen by the user,
				 1 haldane (h), where
				 RF = (1-e(-2m))/2 where m is the map distance
				 2 kosambi (k), where
				 3 e, where the recombination RF is equal to m*/
				map_dist_male = chr[counter].maleCM[counterB+1] - 
chr[counter].maleCM[counterB];
				map_dist_female = chr[counter].femaleCM[counterB+1] - 
chr[counter].femaleCM[counterB];

				if ((strcmp(rec_freq_method,"h"))==0) {//haldane chosen
```

```
                                        chr[counter].RF_to_next_marker_maleCM[counterB] =  (1-exp(-
2*map_dist_male/100))/2;
                                        chr[counter].RF_to_next_marker_femaleCM[counterB] = (1-exp(-
2*map_dist_female/100))/2;
                                        warning_out = "\n*Haldane's mapping function used to calculate
recombination frequency.\n";
                                }
                                else {
                                        if ((strcmp(rec_freq_method,"e"))==0) {
                                                chr[counter].RF_to_next_marker_maleCM[counterB] =
map_dist_male/100;
                                                chr[counter].RF_to_next_marker_femaleCM[counterB] =
map_dist_female/100;
                                                warning_out = "\n*Recomination frequency assumed to be equal
to map distance.\n";
                                        }else {
                                        if ((strcmp(rec_freq_method,"k"))==0) {//kosmabi mapping
                                                chr[counter].RF_to_next_marker_maleCM[counterB] =
(exp(4*(map_dist_male/100))-1)/(2+2*exp(4*(map_dist_male/100)));
                                                chr[counter].RF_to_next_marker_femaleCM[counterB] =
(exp(4*(map_dist_female/100))-1)/(2+2*exp(4*(map_dist_female/100)));;
                                                warning_out = "\n*Kosabmi's mapping function used to
calculate recombination frequency.\n";
                                                }else {//no mapping function chosen, default = haldane
                                                        warning_out = "\n*WARNING: recombination
frequency not specified, Haldane's mapping function used.\n";
                                                        chr[counter].RF_to_next_marker_maleCM[counterB] =
(1-exp(-2*map_dist_male))/2;
                                                        chr[counter].RF_to_next_marker_femaleCM[counterB]
= (1-exp(-2*map_dist_female))/2;
                                                }
                                        }
                                }
                        }

                        /*checking for (counterB=1; counterB<(tempchr.markernumber+1);counterB++)  {
                                cout << "\nTempchr B marker number " << counterB << " name: " <<
tempchrB.markernames[counterB];
                        }*/
        } //for each chr
        cout << warning_out;
}

//Function: open the map file and reading the marker info and the QTL's
/* if changes to the marker info file is to be made it should be doen in this
 function. This includes if the QTL's for some reason are included between the markers*/
bool loadmap(string thefilename){
        //var
        bool opened;
        bool qtls_added = 0; //to only do the qtl adding once
	char linenames[40], transfer[40];//the names of the lines indicating variables
	char inttochar[40];
	int counter, counterB, empty;//count the number of cols when multiple items per line

	char *charconvert = (char*)thefilename.c_str();//conversion of string to char array
	FILE* thefile = fopen(charconvert, "r");

	total_markernumber = 0;

	if (thefile==NULL) {
		cout << "\n****ERROR: No map file with QTL and marker postitions named: '"<< 
thefilename
		<<"' ,please edit main infomation file to indicate correct map file to load.\n";
		opened = false;
			}
	else {//reading the file
		//celcounter = 0; // see how many cells there are in the file
		while(feof(thefile)==0) {//benchmarked the reading of the file on my mac to approx 
20sec/gig
			fscanf(thefile, "%s", linenames);//scan the line names for var indicators

			//almost CASE, which would have been nice here
			//1 CHROMOSOME SECTION
			if ((strcmp(linenames, "chromosome_number"))==0) {
				fscanf(thefile, "%d",&chromosome_number);
				chr = new chromosome[chromosome_number+1];

				//including space for our qtls in the arrays
				if (qtls_added == 0) {
					for (counter=1;counter<(qtl_number+1); counter++) {
						chr[qtl[counter].qtl_chro].qtl_per_chr++;
					}
					qtls_added = 1;
				}


				for (counter = 1; counter<(chromosome_number+1); counter++) {
					fscanf(thefile, "%s%d",chr[counter].chr_name, 
&chr[counter].markernumber);
					//setting the length of maker names and, maps
					//could make this automatic if ness, espessially when creating 
automated txtfiles
					total_markernumber = total_markernumber + 
chr[counter].markernumber;//must be done before changed to include the qtls (NB)
					chr[counter].markernumber = chr[counter].markernumber + 
chr[counter].qtl_per_chr;
					chr[counter].markernames = new 
string[chr[counter].markernumber+1];
					chr[counter].maleCM = new 
float[chr[counter].markernumber+1];
```

```
                                        chr[counter].femaleCM = new
float[chr[counter].markernumber+1];
                                        chr[counter].RF_to_next_marker_maleCM = new
float[chr[counter].markernumber+1];
                                        chr[counter].RF_to_next_marker_femaleCM = new
float[chr[counter].markernumber+1];
                                }
                        }


                        if ((strcmp(linenames, "female_map"))==0) {
                                for (counter=1; counter<(chromosome_number+1); counter++) {
                                        for (counterB=1; counterB<(chr[counter].markernumber +1 -
chr[counter].qtl_per_chr); counterB++) {
                                                fscanf(thefile,
"%d%s%f%f",&empty,transfer,&chr[counter].maleCM[counterB],&chr[counter].femaleCM[counterB]);
                                                chr[counter].markernames[counterB] = transfer;
                                        }
                                }
                        }

                //2 QTL SECTION
                        if ((strcmp(linenames, "qtl_number"))==0) {
                                fscanf(thefile, "%d",&qtl_number);
                                qtl = new qtl_array[qtl_number+1];
                                for (counter=1; counter<(qtl_number+1); counter++) {//giving the names
of the chromosomes
                                        sprintf(inttochar,"%d",counter); //converting int to char
                                        strcpy(qtl[counter].qtl_name, "qtl_"); //assigning string values
                                        strncat(qtl[counter].qtl_name, inttochar,5); //concatenating
strings
                                }
                                 for (counter=1; counter<(qtl_number+1);counter++) {
                                         fscanf(thefile, "%s", linenames);
                                        if ((strcmp(linenames,qtl[counter].qtl_name))==0) {
                                                fscanf(thefile,"%d" "%f"
"%f",&qtl[counter].qtl_chro,&qtl[counter].qtl_male,&qtl[counter].qtl_female);
                                        }
                                }
                        }
                } //reading the file while open

                haplotype_size = qtl_number + total_markernumber;
                spliceqtl(); //now adding the qtl to the array

                fclose(thefile);
                opened = true;
                cout << "\nThe marker and QTL file '";
                cout << thefilename <<"' was read...check the output below to see if the data is
correct...\n";
```

```
	}
	return(opened);
}

//Function: creating empty but correct size arrays for the individuals in a given population
void create_empty_pop(population &which_pop, int popID){
	//var
	int counter;

    which_pop.ind = new individual[which_pop.popsize+1]; //setting the popsize array

	for (counter = 1; counter<(which_pop.popsize+1); counter++) {
		which_pop.ind[counter].subpopID = popID;
		which_pop.ind[counter].haplotypeA = new int[haplotype_size+1]; //setting the haplotype
array
		which_pop.ind[counter].haplotypeB = new int[haplotype_size+1]; //setting the haplotype
array
	}
}

//Function: open the file with the number of founders and their diploid genotypes
/* if changes to the founder datafile to be made it should be doen in this
 function. Important to remember that the QTL's should also have their genotypes in the
 founder info file at their position between the markers*/
bool loadfounders(string thefilename){
	//var
	bool opened;
	bool two_lines_per_marker; //double or single line genotypes, no = 0, yes = 1
	char linenames[40], linenamesB[40];
	int counter, counterB;

	char *charconvert = (char*)thefilename.c_str();//conversion of string to char array
	FILE* thefile = fopen(charconvert, "r");

	if (thefile==NULL) {
		cout << "\n****ERROR: No founder genotypes file named: '"<< thefilename
		<<"' ,please edit main infomation file to indicate correct founder genotypes file to
load.\n";
		opened = false;
	}
	else {//reading the file and creating a founder pop

		strcpy(founderpop.popname, "Founder population");

		while(feof(thefile)==0) {
			fscanf(thefile, "%s", linenames);//scan the line names for var indicators

			if ((strcmp(linenames, "2_lines_per_marker"))==0) {//single lines for genotype
or double, should mostly be double
```

```
					fscanf(thefile, "%s",linenamesB);
					if ((strcmp(linenamesB, "yes"))==0) {
						two_lines_per_marker = true;
					}else {
						two_lines_per_marker = false;
					}
				}

				if ((strcmp(linenames, "number_of_founders"))==0) {//defining the number of 
founders
					fscanf(thefile, "%d",&founderpop.popsize);
			//		founderpop.ind = new individual[founderpop.popsize+1]; //setting the 
popsize array
					create_empty_pop(founderpop, -1);//indicating that this is the founder 
population
				}

				if ((strcmp(linenames, "sex"))==0) {//defining the number of founders
					//would like to know if there is a way to read a whole line...
					for (counter = 1; counter<(founderpop.popsize+1); counter++) {
							fscanf(thefile, "%s", linenames);
							if ((strcmp(linenames, "1"))==0) {/*only ness that the 
males must be one, but here it can be decided by
											changing this IF 
statement whet the males and females in the input
											file should be*/
								founderpop.ind[counter].sex = 1; // 
1=TRUE=MALE
							} else {
								founderpop.ind[counter].sex = 0; // 
0=FALSE=FEMALE
							}
					}
				//two line switch here
				//NB currently the individual genotypes should follow directly after the gender 
identification
					for (counterB = 1; counterB<(haplotype_size+1); counterB++) {
						fscanf(thefile, "%s", linenames);
						for (counter = 1; counter<(founderpop.popsize+1); counter++) {
							fscanf(thefile, 
"%d",&founderpop.ind[counter].haplotypeA[counterB]);
						}
							fscanf(thefile, "%s", linenamesB);
						if  ((strcmp(linenames,linenamesB))==0) {
							for (counter = 1; counter<(founderpop.popsize+1); 
counter++) {
								fscanf(thefile, 
"%d",&founderpop.ind[counter].haplotypeB[counterB]);
							}
						}else { //problem with double line input file EXIT here
						cout << "****ERROR: founder input file,
'"<<thefilename<<"' seem to be incorrect. Genotype data incorrect.\n";
						exit(1);
					}
				}
			}
			}
		fclose(thefile);
		opened = true;
		cout << "\nThe founder genotype file '";
		cout << thefilename <<"' was read...check the output below to see if the data is
correct...\n";
	}
	return(opened);
}

//Function: reading the main information file on how to simulate the population, this should be made by
another program at a later stage
void readmainfile(string thefilename, bool& opened, string& markerfilename, string& founderfilename,
string& parentalfilename){
	//var
	char linenames[40];// the names of the lines indicating variables
	bool simulate_map, simulate_founders, simulate_parentals; //make later global if ness

	char *charconvert = (char*)thefilename.c_str();//conversion of string to char array
	FILE* thefile = fopen(charconvert, "r");


	simulate_parentals = false;
	simulate_founders = false;
	simulate_map = false;

	if (thefile==NULL) {
		cout << "\nNo main info file named: '"<< thefilename
		<< "' , please select another file.\n";
		opened = false;
	}

	else {//reading the file
		while(feof(thefile)==0) {
			fscanf(thefile, "%s", linenames);//scan the line names for var indicators

			if ((strcmp(linenames, "marker_type"))==0) {//marker type
				fscanf(thefile, "%s", linenames);
				marker_type = linenames;
			}

			if ((strcmp(linenames, "create_map"))==0) {//loading the marker and qlt datafile
					fscanf(thefile, "%s",linenames);
```

```
						if ((strcmp(linenames, "no"))==0){//do not simulate but
read file
							simulate_map = false;
							fscanf(thefile, "%s",linenames);
							if ((strcmp(linenames, "load_map_file"))==0){
								fscanf(thefile, "%s",linenames);
								markerfilename = linenames;
							}
						}
				else {//marker map needs to be simulated
					simulate_map = true;
					markerfilename = "SIMULATE_MAP_DATA";
				}
			}

			if ((strcmp(linenames, "create_founders"))==0) {/* loading the parental
genotypes from this file

		should not be linked witht the population size*/
				fscanf(thefile, "%s",linenames);
				if ((strcmp(linenames, "no"))==0){//do not simulate but read file
					simulate_founders = false;
					fscanf(thefile, "%s",linenames);
					if ((strcmp(linenames, "load_founders_genotypes"))==0){
						fscanf(thefile, "%s",linenames);
						founderfilename = linenames;
					}
				}
				else {//marker map needs to be simulated
					simulate_founders = true;
					founderfilename = "SIMULATE_FOUDER_DATA";
				}
			}

			if ((strcmp(linenames, "create_parentals"))==0) {

				fscanf(thefile, "%s",linenames);
				if ((strcmp(linenames, "no"))==0){//do not simulate but read file
					simulate_parentals = false;
					cout << "Founders are parentals: burnin generations = 0";
					} else {
					simulate_parentals = true;
					}
			}

					if ((strcmp(linenames,
"generations_to_parental_generation"))==0) {

					fscanf(thefile, "%d", &burnin_generations);
					}
```

```
                                                if (simulate_parentals == false) {
                                                        burnin_generations = 0;
                                                }



                                                if ((strcmp(linenames, "number_of_parental_populations"))==0)
{
                                                fscanf(thefile, "%d", &subpopnumber);
                                                }

                                                if ((strcmp(linenames,
"individuals_in_parental_populations"))==0) {
                                                fscanf(thefile, "%d", &subpopsize);
                                                }

                                                if ((strcmp(linenames, "recombination_frequency"))==0) {
                                                fscanf(thefile, "%s", rec_freq_method);
                                                }
                                                parentalfilename = "SIMULATE_PARENTAL_DATA";



                }
                fclose(thefile);
                opened = true;
        }
        population_size_total = subpopnumber*subpopsize; //only valid while subpops are the same size
}

//Function: the main reproduction function. This function will hold all the steps to replace the current
population with its offspring
//Used to simulate the Founder populations, if they are not loaded (will unfortunately have some
duplication of this code in the simulate cross function.
void reproduce(int generations) {
        //var
        int counter, counterB, counterC, counterD, counterE, counterF;
        int mateone, matetwo;
        individual par1, par2;
        bool which_haplotype_p1, which_haplotype_p2;
        float chance01;
        int matings_per_subpop, offspring_per_mating;
        int whichind;
        int male_matings, female_matings; //some special variables that indicates how many times (max
value) any individual of a spec sex can mate

        //max matings
        male_matings = population_size_total*2; //max number times a male can mate
        female_matings = male_matings; //max number of times a female can mate
        offspring_per_mating = 1;
```

```
	for (counter = 1; counter<(subpopnumber + 1); counter++) {//for all the subpopulations

			whichind = 0;
			//matings_per_subpop = subpop[counter].popsize;//this is used when only one offspring
per mating are created
			matings_per_subpop = subpop[counter].popsize/offspring_per_mating; //use thi formula
to keep the popsize constant, check that it is an integer

	for (counterB = 1; counterB<(matings_per_subpop+1); counterB++) {//for all individuals the new
populations population


	/*mate selection 1, selecting a male or female at random,
	i.e. no preferecne for any sex to be selected at first (can change this when one or the other sex has
some fitness thing)
	from which pop -- founder (gen ==1 ), same or a migrant */

					mateone = 0;//to reset the first mate
					matetwo = 0;//to reset the 2nd mate




				//PUT THIS IN A FUNCTION AND RETURN THE INDIVUDUAL
PARENTS, make switches for diff stages of generation times and migration
					if (generations == 1) {//for the first generation and from the -----
FOUNDER POPULATION-------
						mateone = rand() % founderpop.popsize + 1;
	/*mate selection 2, selecting second mate. Must be different individual and opposite sex.*/

						do { //must be do while do at least one before continuing
							matetwo = rand() % founderpop.popsize + 1;
						} while ((matetwo == mateone) or (founderpop.ind[mateone].sex
== founderpop.ind[matetwo].sex));

						//parental assignment (to make sure the male is par one
						if (founderpop.ind[mateone].sex == 1) {
						par1 = founderpop.ind[mateone];
						par2  = founderpop.ind[matetwo];
						//	cout << "Subpop: " << counter << ", Individual: " <<
counterB << "\n";
						//	cout << "Mateone " << mateone << " Par 1 Sex: "<<
par1.sex << ", PopID: " << par1.subpopID <<  " , 1st allele: "<< par1.haplotypeA[1] << ", 2nd allele: "<<
par1.haplotypeB[1] << "\n";
						//	cout << "Matetwo " << matetwo << " Par 2 Sex: "<<
par2.sex << ", PopID: " << par2.subpopID <<  " , 1st allele: "<< par2.haplotypeA[1] << ", 2nd allele: "<<
par2.haplotypeB[1] <<  "\n";
						} else {
								par2 = founderpop.ind[mateone];
								par1  = founderpop.ind[matetwo];
					//		cout << "Subpop: " << counter << ", Individual: " <<
counterB << "\n";
					//		cout << "Mateone " << mateone << " Par 1 Sex: "<<
par1.sex << ", PopID: " << par1.subpopID <<  " , 1st allele: "<< par1.haplotypeA[1] << ", 2nd allele: "<<
par1.haplotypeB[1] << "\n";
					//		cout << "Matetwo " << matetwo << " Par 2 Sex: "<<
par2.sex << ", PopID: " << par2.subpopID <<  " , 1st allele: "<< par2.haplotypeA[1] << ", 2nd allele: "<<
par2.haplotypeB[1] <<  "\n";
					}
				} else {//when parrents are from ------SUB POPULATION-----
					//no migration in this function yet

				//----------RANDOM MATING POPULATION SECTION---------
					mateone = rand() % subpop[counter].popsize + 1;
					//mate selection 2, selecting second mate. Must be different
individual and opposite sex.
					do { //must be do while do at least one before continuing
						matetwo = rand() % subpop[counter].popsize + 1;
					} while ((matetwo == mateone) or
(subpop[counter].ind[mateone].sex == subpop[counter].ind[matetwo].sex));

					//parental assignment (to make sure the male is par one
					if (subpop[counter].ind[mateone].sex == 1) {
						par1 = subpop[counter].ind[mateone];
						par2  = subpop[counter].ind[matetwo];
//										cout << "Subpop: " << counter
<< ", Individual: " << counterB << "\n";
//										cout << "Mateone " << mateone
<< " Par 1 Sex: "<< par1.sex << ", PopID: " << par1.subpopID <<  " , 1st allele: "<< par1.haplotypeA[1]
<< ", 2nd allele: "<< par1.haplotypeB[1] << "\n";
//										cout << "Matetwo " << matetwo
<< " Par 2 Sex: "<< par2.sex << ", PopID: " << par2.subpopID <<  " , 1st allele: "<< par2.haplotypeA[1]
<< ", 2nd allele: "<< par2.haplotypeB[1] <<  "\n";
				//		cout << "M1A=P1A M1B=P1B M2A=P2A
M2B=P2B\n";
				//		cout
<<subpop[counter].ind[mateone].haplotypeA[1]<<"="<<par1.haplotypeA[1]<<"
"<<subpop[counter].ind[mateone].haplotypeB[1]<<"="<<par1.haplotypeB[1]<<"
"<<subpop[counter].ind[matetwo].haplotypeA[1]<<"="<<par2.haplotypeA[1]<<"
"<<subpop[counter].ind[matetwo].haplotypeB[1]<<"="<<par2.haplotypeB[1]<<"\n" ;
					} else {
						par2 = subpop[counter].ind[mateone];
						par1  = subpop[counter].ind[matetwo];
//										cout << "Subpop: " << counter
<< ", Individual: " << counterB << "\n";
//										cout << "Mateone " << mateone
<< " Par 1 Sex: "<< par1.sex << ", PopID: " << par1.subpopID <<  " , 1st allele: "<< par1.haplotypeA[1]
<< ", 2nd allele: "<< par1.haplotypeB[1] << "\n";
//                                                                          cout << "Matetwo " << matetwo
<< " Par 2 Sex: "<< par2.sex << ", PopID: " << par2.subpopID <<  " , 1st allele: "<< par2.haplotypeA[1]
<< ", 2nd allele: "<< par2.haplotypeB[1] <<  "\n";
                                  //                  cout << "M2A=P1A M2B=P1B M1A=P2A
M1B=P2B\n";
                                  //                  cout
<<subpop[counter].ind[matetwo].haplotypeA[1]<<"="<<par1.haplotypeA[1]<<"
"<<subpop[counter].ind[matetwo].haplotypeB[1]<<"="<<par1.haplotypeB[1]<<"
"<<subpop[counter].ind[mateone].haplotypeA[1]<<"="<<par2.haplotypeA[1]<<"
"<<subpop[counter].ind[mateone].haplotypeB[1]<<"="<<par2.haplotypeB[1]<<"\n" ;

                                          }
                                          //counting their matings
                                          subpop[counter].ind[mateone].times_mated =
subpop[counter].ind[mateone].times_mated +1;
                                          subpop[counter].ind[matetwo].times_mated =
subpop[counter].ind[matetwo].times_mated +1;

                                  //      cout << mateone << "\n";
                                  //      cout << matetwo << "\n";

                                  //----------RANDOM MATING POPULATION SECTION-- END--------
-*/

                 //SWITCH HERE BETWEEN RANDOM AND STRUCTURED POPULATIONS

                                  /*//----------STRUCTURED POPULATION section-----------------
                                          //this is currently quite inefficient as all the individuals are
chosen by chance
                                          //even the last ones in the population when still unmated, this
will be a problem when the population sizes becomes big


                                          //selecting male first
                                          do {
                                                  mateone = rand() % subpop[counter].popsize + 1;
                                          } while ((subpop[counter].ind[mateone].sex != 1) or
(subpop[counter].ind[mateone].times_mated == male_matings)
                                                           or
(subpop[counter].ind[mateone].selected_to_mate != 1));//only some males can mate

                                          //mate selection 2, selecting FEMALE. .
                                          do { //must be do while do at least one before continuing
                                                  matetwo = rand() % subpop[counter].popsize + 1;
                                          } while ((subpop[counter].ind[matetwo].sex != 0) or
(subpop[counter].ind[matetwo].times_mated == female_matings));

                                  //counting their matings and stopping remating
                                  subpop[counter].ind[mateone].times_mated =
subpop[counter].ind[mateone].times_mated +1;
```

```
					subpop[counter].ind[matetwo].times_mated = 
subpop[counter].ind[matetwo].times_mated +1;

						par1 = subpop[counter].ind[mateone];
						par2  = subpop[counter].ind[matetwo];
					//----------STRUCTURED POPULATION section-- end-----------*/

					}

	/*mating -- crossing over and independent assortment & creating offspring pops
	 since the migration will be done by selecting parents from different populations
	 this section do not need to account for that, this means the new individuals are
	 placed in their correct  places (see counter and counterB)*/


	for (counterF = 1; counterF<(offspring_per_mating+1);counterF++) {//repeat for the number of 
matings
	 counterE = 0;//for deriving the marker number from chr number and marker nr on the chr.

		whichind = whichind + 1;
		subpop_offspring[counter].ind[whichind].father = mateone;
		subpop_offspring[counter].ind[whichind].mother = matetwo;
		//cout << "pop: " << counter << " counterB: " << counterB << " counterF:  " << counterF 
<< " the individual: " << whichind << "\n";

	for (counterC = 1; counterC<(chromosome_number +1); counterC++) {//per chr?

		which_haplotype_p1 = 0;
		which_haplotype_p2 = 0;

		if (rand()%2 == 1) {
			which_haplotype_p1 = 1;

		}/*cout << "1\n";
			} else {
				cout << "0\n";
			}*/


		if (rand()%2 == 1) {
			which_haplotype_p2 = 1;
		}

		for (counterD=1; counterD<(chr[counterC].markernumber +1); counterD++) {//for each 
of the markers
			counterE = counterE + 1;

			/*/All loci unlinked experiment FOR TESTING
			/////////////////remove after experiment, here all the loci are 
unlinked!!!//////////////////
```

```
				which_haplotype_p1 = 0;///////////
				which_haplotype_p2 = 0;////////////
				//////////
				if (rand()%2 == 1) {//////////
					which_haplotype_p1 = 1;//////////
				}//////////////
				if (rand()%2 == 1) {/////////
					which_haplotype_p2 = 1;/////////
				}/////////////////remove after experiment, here all the loci are
unlinked!!!//////////////*/


				//RANDOMLY from male or female parent first
				//if (rand()%2 == 1) {//Chromosome from the male first

				//from parent 1 the male parent
				if (which_haplotype_p1 == 1) {//haplotypeA from par_1 markers will be
allocated to the new individual's A hapl
					subpop_offspring[counter].ind[whichind].haplotypeA[counterE] =
par1.haplotypeA[counterE];
//					cout << "(a)Marker number : "<< (counterE)<<"\n";
				}
				else {//haplotypeB from par_1 will be allocated to the new individual's A hapl
					subpop_offspring[counter].ind[whichind].haplotypeA[counterE] =
par1.haplotypeB[counterE];
//					cout << "(b)Marker number : "<< (counterE)<<"\n";
				}
				/*CROSSINGOVER HERE for the male parent*/
				chance01 = (rand()%100001);
				chance01 = chance01/100000;
				if (chance01 <= chr[counterC].RF_to_next_marker_maleCM[counterD]) {
					which_haplotype_p1 =  !which_haplotype_p1;
//					cout << "switch (male): chance " << chance01 << " <= recfreq "<<
chr[counterC].RF_to_next_marker_maleCM[counterD];
//					cout << "\nwhichhap = " << which_haplotype_p1 <<"\n";
				}
				//male end

			//from parent 2, female parent
				if (which_haplotype_p2 == 1) {//haplotypeA from par2 markers will be allocated
to the new individual's A hapl
					subpop_offspring[counter].ind[whichind].haplotypeB[counterE] =
par2.haplotypeA[counterE];
//					cout << "(a)Marker number : "<< (counterE)<<"\n";
				}
				else {//haplotypeB from par2 will be allocated to the new individual's A hapl
					subpop_offspring[counter].ind[whichind].haplotypeB[counterE] =
par2.haplotypeB[counterE];
//					cout << "(b)Marker number : "<< (counterE)<<"\n";
				}
```

```
                                        /*CROSSINGOVER HERE female*/
                                        chance01 = (rand()%100001);
                                        chance01 = chance01/100000;
                                        if (chance01 <= chr[counterC].RF_to_next_marker_femaleCM[counterD]) {
                                                which_haplotype_p2 =  !which_haplotype_p2;


//                                              cout << "switch (fem): chance " << chance01 << " <= recfreq "<< 
chr[counterC].RF_to_next_marker_femaleCM[counterD];
//                                              cout << "\nwhichhap = " << which_haplotype_p2 <<"\n";
                                        }//female end.
                                        //}                             //-------------swithc to chromosomeA from female first--
------------
                                /*      else {
                                                //from parent 2 the female parent
                                                if (which_haplotype_p2 == 1) {//haplotypeA from par_2 markers will be 
allocated to the new individual's A hapl
                                                        subpop_offspring[counter].ind[whichind].haplotypeA[counterE] 
= par2.haplotypeA[counterE];
                                                                cout << "(a)Marker number : "<< (counterE)<<"\n";
                                                }
                                                else {//haplotypeB from par_2 will be allocated to the new individual's A 
hapl
                                                        subpop_offspring[counter].ind[whichind].haplotypeA[counterE] 
= par2.haplotypeB[counterE];
                                                                cout << "(b)Marker number : "<< (counterE)<<"\n";
                                                }
                                                //CROSSINGOVER HERE for the female parent
                                                chance01 = (rand()%100001);
                                                chance01 = chance01/100000;
                                                if (chance01 <= 
chr[counterC].RF_to_next_marker_femaleCM[counterD]) {
                                                        which_haplotype_p2 =  !which_haplotype_p2;
                                                }
                                                //female end

                                                //from parent 1, male parent
                                                if (which_haplotype_p1 == 1) {//haplotypeA from par1 markers will be 
allocated to the new individual's A hapl
                                                        subpop_offspring[counter].ind[whichind].haplotypeB[counterE] 
= par1.haplotypeA[counterE];
                                                                cout << "(a)Marker number : "<< (counterE)<<"\n";
                                                }
                                                else {//haplotypeB from par1 will be allocated to the new individual's A 
hapl
                                                        subpop_offspring[counter].ind[whichind].haplotypeB[counterE] 
= par1.haplotypeB[counterE];
                                                                cout << "(b)Marker number : "<< (counterE)<<"\n";
                                                }
                                                //CROSSINGOVER HERE male
                                        chance01 = (rand()%100001);
                                        chance01 = chance01/100000;
                                        if (chance01 <= chr[counterC].RF_to_next_marker_maleCM[counterD])
{
                                                which_haplotype_p1 =  !which_haplotype_p1;
                                        }//male end.
                                } */ //end of if else statement that chooce which parent will donate their alleles
first

                         }//number of markers per chr end

                        }//chr end
                        }//number off offspring per mating loop (counter F)
                  }
                }

//----------- here all the new individuals are created -----------------

                //mutation here if ness, different types for different markers

                //showing the old population before replacement with the new stuff
//              for (counter = 1; counter<(subpopnumber + 1); counter++){
//                      showpop(subpop[counter]);
//              }

                //replacing parental populations and resetting the other values
                        for (counter = 1; counter<(subpopnumber + 1); counter++) {//for all the subpopulations
                                for (counterB = 1; counterB<(subpop[counter].popsize+1); counterB++) {//for all
individuals in a population
                                        for (counterC = 1; counterC<(haplotype_size+1); counterC++) {
                                                subpop[counter].ind[counterB].haplotypeA[counterC] =
subpop_offspring[counter].ind[counterB].haplotypeA[counterC];
                                                subpop[counter].ind[counterB].haplotypeB[counterC] =
subpop_offspring[counter].ind[counterB].haplotypeB[counterC];
                                        }
                                        subpop[counter].ind[counterB].mother =
subpop_offspring[counter].ind[counterB].mother;
                                        subpop[counter].ind[counterB].father =
                subpop_offspring[counter].ind[counterB].father;
                                        subpop[counter].ind[counterB].subpopID = counter;

                                        subpop[counter].ind[counterB].times_mated = 0;
                                        subpop[counter].ind[counterB].phenotype = 0;
                                        subpop[counter].ind[counterB].phenotype2 = 0;
                                //      subpop[counter].ind[counterB].selected_to_mate = 0;
                                        /*forced-exact sex ratio
                                        to get the correct sex ratio here the following must be done
                                        take the sexratio, i.e. = nr_males/total_population_size
                                         and invert this, i.e. =  *x-1 */
                                //also the selected to mate parameter must be one to enable mating
```

```
					if (counterB % 2==0) {//the modulus operator
						subpop[counter].ind[counterB].sex = 1;//male
						subpop[counter].ind[counterB].selected_to_mate = 1;

					}
					else {
						subpop[counter].ind[counterB].sex = 0;//female
						subpop[counter].ind[counterB].selected_to_mate = 1;
					}
			}
	}
	//showing fixations, heterozygosity,
}

//function: The first selection type that can be done
void selection_1() {
	//var
	int chance, counter, counterB;
	int mating_males;
	//int mating_females; // the number of males and females that can mate

	mating_males = 14;
	//mating_females = 5; currently all the females can mate see above

	//random selection of a number of males and females to mate
	for (counter = 1; counter<(subpopnumber + 1); counter++) {//for all the subpopulations

			//for selecting the males
			for (counterB = 1;counterB<(mating_males+1); counterB++) {//enabeling the number of 
males to mate
				do {//redraw ind if female or allready selected
					chance = (rand()%(subpop[counter].popsize + 1));
				}	while ((subpop[counter].ind[chance].sex == 0) or 
(subpop[counter].ind[chance].selected_to_mate == 1));
				subpop[counter].ind[chance].selected_to_mate = 1; //now this male can mate
				//cout << "\n male enabled counterB " <<  counterB;
			}
	}

}

//function: to record the information for some of the generations
void record_info_for_gen(int which_gen) {
	//var
	int counter, counterB, counterC;
	int *allele_type_holder;
	bool same_holder, tot_holder;
	int sexratio_o;

	//setting the array for each subpop in this generation array
```

```
		info_per_gen_per_pop[which_gen] = new saved_information[subpopnumber+1];
		//the holder array
		allele_type_holder = new int[haplotype_size+1];
		//the allel freq array
		for (counter = 1; counter<(subpopnumber+1); counter++) {
			info_per_gen_per_pop[which_gen][counter].one_allele_freq = new
int[haplotype_size+1];
		}

		//cout << "Generation: "<<which_gen<<"\n";

		sexratio_o = 0;
		for (counter = 1; counter<(subpopnumber+1); counter++) {//for all the subpops
			for (counterB = 1; counterB<(haplotype_size+1); counterB++) {//for all the loci

				/*NOTE: ALLELE FREQ
				 the next line is an enigma, it is used to record the allels frequency per locus per
subpop
					BUT
				it uses the fist allele of the first individual (for all loci) as the allele type to
compare all others to
				this in itself is not a problem but it does make generation wise following of a
allele frequency dificult using this
				 recorded value as it will randomly assign allele types per generation to the allele
type when there is more than one allele
				 in the population.
				 THEREFORE, dont use this when trying to follow the allele frequencies for
each generation!!
				 unless the holder's type is set to an allele value that could be found in the
population BUT then the fixation per total population cant
				 be calculated
				 */
				allele_type_holder[counterB] =
subpop[counter].ind[1].haplotypeA[counterB];//to set the holder's first alleletype

				info_per_gen_per_pop[which_gen][counter].one_allele_freq[counterB] = 1;
				if (subpop[counter].ind[1].haplotypeB[counterB] ==
allele_type_holder[counterB]) {//continue to add dipl allelel same as 1st
					info_per_gen_per_pop[which_gen][counter].one_allele_freq[counterB] =
info_per_gen_per_pop[which_gen][counter].one_allele_freq[counterB] + 1;
				}
			}
			//note this is for the first male as the next loop is from ind 2 onwards
			if (subpop[counter].ind[1].sex == 1) {//male
				sexratio_o = sexratio_o + 1;
			}

			for (counterB = 2/*NOTE THIS "2"*/; counterB<(subpop[counter].popsize+1);
counterB++) {//for the ind per pop FROM ind 2
				if (subpop[counter].ind[counterB].sex == 1) {//male
```

```
					sexratio_o = sexratio_o + 1;
				}
				for (counterC = 1; counterC<(haplotype_size+1); counterC++) { //remember, 
they are diploid
					if (subpop[counter].ind[counterB].haplotypeA[counterC] == 
allele_type_holder[counterC]) {//continue to add if same as 1st

	info_per_gen_per_pop[which_gen][counter].one_allele_freq[counterC] = 
info_per_gen_per_pop[which_gen][counter].one_allele_freq[counterC] + 1;
					}
					if (subpop[counter].ind[counterB].haplotypeB[counterC] == 
allele_type_holder[counterC]) {//continue to add if same as 1st for allele B

	info_per_gen_per_pop[which_gen][counter].one_allele_freq[counterC] = 
info_per_gen_per_pop[which_gen][counter].one_allele_freq[counterC] + 1;
					}
				}

				}//end of ind loop

			/*Recording the info for each subpopulation here*/
			info_per_gen_per_pop[which_gen][counter].prop_fixed_loci = 0;
			for (counterC = 1; counterC<(haplotype_size+1); counterC++) {
				if (info_per_gen_per_pop[which_gen][counter].one_allele_freq[counterC] == 
(subpop[counter].popsize*2)) {//counting the no of fixed loci
					info_per_gen_per_pop[which_gen][counter].prop_fixed_loci = 
info_per_gen_per_pop[which_gen][counter].prop_fixed_loci + 1;
				}

			}

			//testout
	//		cout << "SubPop: " << counter << " (popsize = "<<subpop[counter].popsize<<")\n";
	//		cout << "Males: "<< sexratio_o << ", (no males recoreded): "<< 
info_per_gen_per_pop[which_gen][counter].sexratio <<"\n";
	//		cout << "Number_fixed_loci: "<< 
info_per_gen_per_pop[which_gen][counter].prop_fixed_loci <<"\n";
	//		cout << "Marker_number_(of_the_first_allele): ";
	//		for (counterC = 1;counterC<(haplotype_size+1); counterC++) {
	//			cout << info_per_gen_per_pop[which_gen][counter].one_allele_freq[counterC] 
<< " ";
	//		}
	//		cout << "\n";

		}//end of pop

		/*Recording the information per total pop*///this could maybe be moved up into the prev loop
		same_fixed_loci[which_gen] = 0;
		tot_fixed_loci[which_gen] = 0;
			//this could mybe be better by reducing the else statements, think a bit
	for (counter = 1; counter<(haplotype_size+1); counter++) {//loop other way around...
		same_holder = 1;
		tot_holder = 1;
		for (counterB = 2; counterB<(subpopnumber+1); counterB++) {
			if ((info_per_gen_per_pop[which_gen][counterB-1].one_allele_freq[counter] ==
(subpop[counterB].popsize*2)) and
				(info_per_gen_per_pop[which_gen][counterB].one_allele_freq[counter]
== (subpop[counterB].popsize*2)) and
			   (tot_holder == 1)){
				tot_holder=1;
				if ((subpop[counterB-1].ind[1].haplotypeA[counter] ==
subpop[counterB].ind[1].haplotypeA[counter]) and
					(same_holder == 1)) {//using the 1st ind allele if fixed (will be
fine)
					same_holder = 1;
				}else {
					same_holder = 0;
				}
			}else {
				tot_holder = 0;
				same_holder = 0;
			}
		}
		tot_fixed_loci[which_gen] = tot_fixed_loci[which_gen] + tot_holder;
		same_fixed_loci[which_gen] = same_fixed_loci[which_gen] + same_holder;
		sexratio[which_gen] = sexratio_o;
	}

	//testoutII
/*	cout << "Loci fixed in whole population: " << tot_fixed_loci[which_gen] << "\n";
	cout << "Loci fixed for the same alleles between sub pops: "<< same_fixed_loci[which_gen] <<
"\n";
	cout << "Loci fixed but for different allelels between subpops: "<< tot_fixed_loci[which_gen]-
same_fixed_loci[which_gen] << "\n";
 */
}

//function: Noise Generator
void noisegenerator() {
	//var
	int dis_range, dis_mean;
	int subset_size;
	int counter,counterB;
	div_t devider;
	float sum_of_sq;
	int noisenumber;

	dis_range = 10000;
	devider = div (dis_range,2);
	dis_mean = devider.quot;
	subset_size = 100;
	sum_of_sq = 0;
	noisenumber = population_size_total*2;

	int_noise = new float[noisenumber + 1];

	for (counter=1; counter<(noisenumber+1); counter++) {
		int_noise[counter] = 0;
		for (counterB=1; counterB<(subset_size+1); counterB++) {
			int_noise[counter] = int_noise[counter] + ((rand()%dis_range) + 1);
		}
		int_noise[counter] = (int_noise[counter]/subset_size)-dis_mean;
		sum_of_sq = sum_of_sq + pow((int_noise[counter]/* REMEMBER THE MEAN IS
REMOVED ONE LINE ABOVE -dis_mean*/), 2);
	}

	//Scale HERE
	noise_scale = sqrt(sum_of_sq/noisenumber); //currently this is the standard deviation. Before
changing this THINK FIRST!!
	//cout << "NOISE SCALE !!! " << noise_scale;
}


//PHENOTYPING SUBFUNCTION 1: main effects, single qtl
//format: which individual, which QTL, effect sizes
float single_QTL(individual ind_who, int QTLA, float AA, float Aa, float aa){
	//var
	float phenoval;

	/*NOTE: using else is slightly better since it will often not go to the last
	 class. However a class satement would be much better but cant find any*/

	//00
	if ((ind_who.haplotypeA[qtl[QTLA].qtl_pos] == 0) and
(ind_who.haplotypeB[qtl[QTLA].qtl_pos] == 0)){
		phenoval = AA;
	}
	//01 or 01
	else if (((ind_who.haplotypeA[qtl[QTLA].qtl_pos] == 1) and
(ind_who.haplotypeB[qtl[QTLA].qtl_pos] == 0)) or
			((ind_who.haplotypeA[qtl[QTLA].qtl_pos] == 0) and
(ind_who.haplotypeB[qtl[QTLA].qtl_pos] == 1))){
		phenoval = Aa;
	}
	//11
	else if	((ind_who.haplotypeA[qtl[QTLA].qtl_pos] == 1) and
(ind_who.haplotypeB[qtl[QTLA].qtl_pos] == 1)){
		phenoval = aa;
	}
	return(phenoval);
}

//PHENOTYPING SUBFUNCTION 2: Two way interaction
//format: which individual, which QTL, effect sizes
float two_way(individual ind_who, int QTLA, int QTLB,		float AABB, float AABb, float AAbb,

	float AaBB, float AaBb, float Aabb,

	float aaBB, float aaBb, float aabb){
	//var
	float phenoval;
	string geno2d = "0000";

	//get genotype string
	if (ind_who.haplotypeA[qtl[QTLA].qtl_pos] == 1){geno2d[0] = '1';}
	if (ind_who.haplotypeB[qtl[QTLA].qtl_pos] == 1){geno2d[1] = '1';}
	if (ind_who.haplotypeA[qtl[QTLB].qtl_pos] == 1){geno2d[2] = '1';}
	if (ind_who.haplotypeB[qtl[QTLB].qtl_pos] == 1){geno2d[3] = '1';}

	//get phenotype
	if (geno2d == "0000") {
		phenoval = AABB;
	}
	if ((geno2d == "0001") or (geno2d == "0010")) {
		phenoval = AABb;
	}
	if (geno2d == "0011") {
		phenoval = AAbb;
	}
	//---
	if ((geno2d == "0100") or (geno2d == "1000")) {
		phenoval = AaBB;
	}
	if ((geno2d == "0101") or (geno2d == "0110") or (geno2d == "1001") or (geno2d == "1010")) {
		phenoval = AaBb;
	}
	if ((geno2d == "0111") or (geno2d == "1011")) {
		phenoval = Aabb;
	}
	//---
	if (geno2d == "1100") {
		phenoval = aaBB;
	}
	if ((geno2d == "1101") or (geno2d == "1110")) {
		phenoval = aaBb;
	}
	if (geno2d == "1111") {
		phenoval = aabb;
	}
	return(phenoval);
}

//PHENOTYPING SUBFUNCTION 2: Three way interaction
//format: which individual, which QTL, effect sizes
float three_way(individual ind_who, int QTLA, int QTLB, int QTLC,	float AABBCC, float AABBCc, 
float AABBcc,

				float AABbCC, float AABbCc, float AABbcc,

				float AAbbCC, float AAbbCc, float AAbbcc,


				float AaBBCC, float AaBBCc, float AaBBcc,

				float AaBbCC, float AaBbCc, float AaBbcc,

				float AabbCC, float AabbCc, float Aabbcc,


				float aaBBCC, float aaBBCc, float aaBBcc,

				float aaBbCC, float aaBbCc, float aaBbcc,

				float aabbCC, float aabbCc, float aabbcc){
	//var
	float phenoval = 111111;
	string geno3d = "000000";

	//get genotype string
	if (ind_who.haplotypeA[qtl[QTLA].qtl_pos] == 1){geno3d[0] = '1';}
	if (ind_who.haplotypeB[qtl[QTLA].qtl_pos] == 1){geno3d[1] = '1';}
	if (ind_who.haplotypeA[qtl[QTLB].qtl_pos] == 1){geno3d[2] = '1';}
	if (ind_who.haplotypeB[qtl[QTLB].qtl_pos] == 1){geno3d[3] = '1';}
	if (ind_who.haplotypeA[qtl[QTLC].qtl_pos] == 1){geno3d[4] = '1';}
	if (ind_who.haplotypeB[qtl[QTLC].qtl_pos] == 1){geno3d[5] = '1';}

	//get phenotype
	if (geno3d == "000000") {
		phenoval = AABBCC;
	}
	if ((geno3d == "000001") or (geno3d == "000010")) {
		phenoval = AABBCc;
	}
	if (geno3d == "000011") {
		phenoval = AABBcc;
	}
	//---
	if ((geno3d == "000100") or (geno3d == "001000")) {
			phenoval = AABbCC;
		}
		if ((geno3d == "000101") or (geno3d == "000110") or (geno3d == "001001") or (geno3d ==
"001010")) {
			phenoval = AABbCc;
		}
		if ((geno3d == "000111") or (geno3d == "001011")) {
			phenoval = AABbcc;
		}
		//---
		if (geno3d == "001100") {
			phenoval = AAbbCC;
		}
		if ((geno3d == "001101") or (geno3d == "001110")) {
			phenoval = AAbbCc;
		}
		if (geno3d == "001111") {
			phenoval = AAbbcc;
		}

		//---------
		if ((geno3d == "010000") or (geno3d == "100000")) {
			phenoval = AaBBCC;
		}
		if ((geno3d == "010001") or (geno3d == "010010") or (geno3d == "100001") or (geno3d ==
"100010")) {
			phenoval = AaBBCc;
		}
		if ((geno3d == "010011") or (geno3d == "100011")) {
			phenoval = AaBBcc;
		}
		//---
		if ((geno3d == "010100") or (geno3d == "011000") or (geno3d == "100100") or (geno3d ==
"101000")) {
			phenoval = AaBbCC;
		}
		if ((geno3d == "101010") or (geno3d == "101001") or (geno3d == "100110") or (geno3d ==
"100101") or (geno3d == "011010") or (geno3d == "011001") or (geno3d == "010110") or (geno3d ==
"010101")) {
			phenoval = AaBbCc; //hehe
		}
		if ((geno3d == "010111") or (geno3d == "011011") or (geno3d == "100111") or (geno3d ==
"101011")) {
			phenoval = AaBbcc;
		}
		//---
		if ((geno3d == "101100") or (geno3d == "011100")) {
			phenoval = AabbCC;
		}
```

```
	if ((geno3d == "011101") or (geno3d == "011110") or (geno3d == "101101") or (geno3d ==
"101110")) {
		phenoval = AabbCc;
	}
	if ((geno3d == "101111") or (geno3d == "011111")) {
		phenoval = Aabbcc;
	}

	//---------
	if (geno3d == "110000") {
		phenoval = aaBBCC;
	}
	if ((geno3d == "110001") or (geno3d == "110010")) {
		phenoval = aaBBCc;
	}
	if (geno3d == "110011") {
		phenoval = aaBBcc;
	}
	//---
	if ((geno3d == "110100") or (geno3d == "111000")) {
		phenoval = aaBbCC;
	}
	if ((geno3d == "110101") or (geno3d == "110110") or (geno3d == "111001") or (geno3d ==
"111010")) {
		phenoval = aaBbCc;
	}
	if ((geno3d == "110111") or (geno3d == "111011")) {
		phenoval = aaBbcc;
	}
	//---
	if (geno3d == "111100") {
		phenoval = aabbCC;
	}
	if ((geno3d == "111101") or (geno3d == "111110")) {
		phenoval = aabbCc;
	}
	if (geno3d == "111111") {
		phenoval = aabbcc;
	}
	return(phenoval);
}

//function: phenotyping. This function assign phenotypes, depending on the QTL.
//Important function this NB, may need to move up
//ALSO CANT DO THE FOUNDER POPULATION!!
void phenotyping(){
	//var
	int counter, counterB, counterC;
	float pheno, pheno2;
	float phenoval, phenoval2;
	//phenoval = 10;
	//phenoval2 = 1;

	float noise_inflation;
	noise_inflation = 316227.766; //1 is one sdev, 2 is two sdev ect.


	//note the noise generator generates noise specifically for the number of inds in the pop in the
final gen
	//for now it is fine but will need to be moved and slightly addapted if phenotypes for each gen is
req (only to speed up things)

	/*notes on the phenoytpe generators: the order of the genotype effects are always as follows
	Where caps refer to 0 and smallcaps refer to 1
	single main effects: AA,Aa,aa

  two way interactions: AABB, AABb, AAbb, AaBB, AaBb, Aabb, aaBB, aaBb, aabb

  three way interaction: AABBCC, AABBCc, AABBcc,   AABbCC, AABbCc, AABbcc,   AAbbCC,
AAbbCc, AAbbcc,
							AaBBCC, AaBBCc, AaBBcc,   AaBbCC, AaBbCc,
AaBbcc,   AabbCC, AabbCc, Aabbcc,
							aaBBCC, aaBBCc, aaBBcc,   aaBbCC, aaBbCc,
aaBbcc,   aabbCC, aabbCc, aabbcc,
	 */

	noisegenerator();
	counterC = 1;
	for (counter = 1; counter < (subpopnumber+1); counter++) {//per subpop
		for (counterB = 1; counterB < (subpop[counter].popsize +1); counterB++) {//per ind
			//resetting the pheno var
			pheno = 0;
			pheno2 = 0;

			//here are the main effects for:
			//Q1
		//pheno = pheno + single_QTL(subpop[counter].ind[counterB], 6,1000,0,0);
		//	pheno = pheno + single_QTL(subpop[counter].ind[counterB], 9, 10,10,0);

		//	pheno2 = pheno2 + single_QTL(subpop[counter].ind[counterB], 6, 1,0.5,0);
		//	pheno2 = pheno2 + single_QTL(subpop[counter].ind[counterB], 9, 1,1,0);
			//Q2
			//pheno = pheno + single_QTL(subpop[counter].ind[counterB], 2, 1,2,3);

			//here are the two way interaction for:
			//Q1 and Q2
			pheno = pheno + two_way(subpop[counter].ind[counterB],
6,9,500,0,500,0,1000,0,500,0,500);

			//here are the three way interaction for:
				//Q1 and Q2 and Q3
				//pheno = pheno + three_way(subpop[counter].ind[counterB],
2,6,9,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1000,0,0,0,0,0,0,0,0,0,0,0,0);

				//pheno2 = pheno2 + three_way(subpop[counter].ind[counterB], 2,4,10,
phenoval2,phenoval2,0,0,0,0,(phenoval2-2.0),(phenoval2-2.0),0,(phenoval2-0.5),(phenoval2-
0.5),0,0,0,0,(phenoval2-1.5),(phenoval2-1.5),0,0,0,0,0,0,0,0,0,0,0);

				//ONLY ADDING NOISE (the int_noise array has a noise parameter for each
individual)

				pheno = pheno + int_noise[counterC]/noise_scale*(sqrt(noise_inflation)); //see
noise scale above
				counterC++;

				pheno2 = pheno2 + int_noise[counterC]/noise_scale*(sqrt(noise_inflation));//see
noise scale above
				counterC++;

				//Adding the phenotype to the ind
				subpop[counter].ind[counterB].phenotype = pheno;
				subpop[counter].ind[counterB].phenotype2 = 0;//pheno2;
			}
	}

	//will do this for all the individuals in the whole population.
}

//function: this function is to record/write some information IN A FILE
void outputfunction(int which_gen_rec) {
	//var
	FILE * outfile;
	char outfilename[256] = {0};
	char gennum[5];
	int counter, counterB, counterC, counterD;

	strcat(outfilename,"Generation_");
	sprintf(gennum,"%d",which_gen_rec);
	strcat(outfilename, gennum);
	strcat(outfilename,"_genotypes.txt");

	//cout << "\nQTL array data TEST\n";
	//for (counter = 1; counter< (qtl_number+1); counter++) {
	//	cout << counter <<" \t";
	//	cout << qtl[counter].qtl_pos << "\n";
	//}

	outfile = fopen(outfilename,"w");
//	fprintf(outfile,"Generation, %d",which_gen_rec);
	fprintf(outfile, "Individual\tPopulation\tSex\tFather\tMother\tPhenotype\tPhenotype2");
	for (counter = 1; counter < (qtl_number +1 ); counter++) { //writing the QTLs first in the format 
Qnumber,positio in haplotype
		fprintf(outfile, "\tQ%d_%d", counter, qtl[counter].qtl_pos);
	 //   fprintf(outfile, "\tQ%d,%d", counter, qtl[counter].qtl_pos);
	}

	counterD = 1;
	for (counter = 1; counter<(haplotype_size+1); counter++) { //writing the names excluding the 
QTls
			if (counter == (qtl[counterD].qtl_pos)) {
				counterD = counterD + 1;
			}else {
				fprintf(outfile,"\t%d",counter-(counterD)+1);
		//			fprintf(outfile,"\t%d",counter);
			}
	}

	/* Cant print burnin Gen before fixed
	 if (burnin_generations == 0) {//founder generation

		for (counterB = 1; counterB < (founderpop.popsize +1); counterB++) {//per ind
			fprintf(outfile, 
"\n%d\t%d\t%d\t%d\t%d\t%f\t%f",counterB,founderpop.ind[counterB].subpopID,founderpop.ind[counter
B].sex,0,0,founderpop.ind[counterB].phenotype,founderpop.ind[counterB].phenotype2);
			for (counterC=1; counterC < (qtl_number +1 ); counterC++) {
				fprintf(outfile, 
"\t%d%d",founderpop.ind[counterB].haplotypeA[qtl[counterC].qtl_pos],founderpop.ind[counterB].haplot
ypeB[qtl[counterC].qtl_pos]);
			}
			counterD = 1;
			for (counterC=1; counterC < (haplotype_size+1); counterC++ ) {
				if (counterC == (qtl[counterD].qtl_pos)) {
					counterD = counterD + 1;
				}else {
					fprintf(outfile, 
"\t%d%d",founderpop.ind[counterB].haplotypeA[counterC],founderpop.ind[counterB].haplotypeB[count
erC]);
				}
			}
		}
	}else
	 */
	//non founder generation
		for (counter = 1; counter < (subpopnumber+1); counter++) {//per subpop
			for (counterB = 1; counterB < (subpop[counter].popsize +1); counterB++) {//per 
ind
				fprintf(outfile, 
"\n%d\t%d\t%d\t%d\t%d\t%f\t%f",counterB,subpop[counter].ind[counterB].subpopID,subpop[counter].in
d[counterB].sex,subpop[counter].ind[counterB].father,subpop[counter].ind[counterB].mother,subpop[cou
nter].ind[counterB].phenotype,subpop[counter].ind[counterB].phenotype2);
```

```
					for (counterC=1; counterC < (qtl_number +1 ); counterC++) {
						fprintf(outfile, 
"\t%d%d",subpop[counter].ind[counterB].haplotypeA[qtl[counterC].qtl_pos],subpop[counter].ind[counter
B].haplotypeB[qtl[counterC].qtl_pos]);

					}
					counterD = 1;
					for (counterC=1; counterC < (haplotype_size+1); counterC++ ) {
						if (counterC == (qtl[counterD].qtl_pos)) {
							counterD = counterD + 1;
						}else {
							if (subpop[counter].ind[counterB].haplotypeA[counterC] 
== 0 and subpop[counter].ind[counterB].haplotypeB[counterC] == 0) {
								fprintf(outfile, "\t%d", 1);
							}
							if (subpop[counter].ind[counterB].haplotypeA[counterC] 
== 1 and subpop[counter].ind[counterB].haplotypeB[counterC] == 0) {
								fprintf(outfile, "\t%d", 2);
							}
							if (subpop[counter].ind[counterB].haplotypeA[counterC] 
== 0 and subpop[counter].ind[counterB].haplotypeB[counterC] == 1) {
								fprintf(outfile, "\t%d", 2);
							}
							if (subpop[counter].ind[counterB].haplotypeA[counterC] 
== 1 and subpop[counter].ind[counterB].haplotypeB[counterC] == 1) {
								fprintf(outfile, "\t%d", 3);
							}
						//	fprintf(outfile, 
"\t%d%d",subpop[counter].ind[counterB].haplotypeA[counterC],subpop[counter].ind[counterB].haplotyp
eB[counterC]);

						}
					}
				}
	}

	fclose(outfile);
}


//THE MAIN PROGRAM
int main (int argc, char * const argv[]) {
	//LOCAL VARIABLES
	string xfile,markerfile,founderfile,parentalfile;//the file name that needs to be opened
	char inttochar[40];
	bool exitnow; //use when quitting is ness
	bool xfileopen;//could the file be read correctly
	char switch_questions[256];
	int counter;
	//int counterB;
	//int record_fraction;//the number of generations in the simulation that info will be recorded for


	//BEGIN
	srand (time(NULL)); //making the randomizer slightly more random but can change to have a 
seeding value...weird but true
	system("clear");//clear the terminal
	cout<< "Running QTL simulator v1 (Ronald Nelson):\n";


	//getting correct file to open (the showfile and openfunction would be better if together)
	xfileopen = false;
	exitnow = false;
	while ((exitnow==false) and (xfileopen==false)) {
		xfile = showfiles();//point to the main file to be opened
			if ((xfile == "q") or (xfile == "Q")) {
					exitnow = true; //quitting the progam
				}
				else {
					readmainfile(xfile,exitnow,markerfile,founderfile,parentalfile);

				}
	}//to repeat until the correct file is opend or the user quits


	if (markerfile == "SIMULATE_MAP_DATA") { //loading or creating marker and qtl info file
		cout << "\nSimulating founder dataset...\n";
		//make file according to user input, and save file
		//xfileopen = open the just created datafile
	}
	else {
		if (markerfile != "") {
			xfileopen = loadmap(markerfile);//opening and reading the markers and qtls
		}
	}


	if (founderfile == "SIMULATE_FOUDER_DATA") { //loading or creating the founder 
genotypes if individual....
		cout << "\nSimulating founder dataset...\n";
		//make file according to user input, and save file
		//xfileopen = open the just created datafile
	}
	else {
		if (founderfile != "") {
			xfileopen = loadfounders(founderfile);//opening and reading the markers and qtls
		}
	}
```

```
        if (xfileopen==true){//only continue the program from here if the MARKER- and FOUNDER-
files was opened sucessfully
    show_marker_info();//numerical display of data
            plot_marker_info();//graphical display of data
            showpop(founderpop);


            if (parentalfile == "SIMULATE_PARENTAL_DATA") //----switch here between
loading parentals or creating parentals
                    {//-----------------------START TO simulate parentals...default-----------------------

                        cout << "Continue to simulate "<< subpopnumber <<" sub-populations
from founder population for " << burnin_generations <<  " generations to create parental lines (Y/N).\n";
                        cin.getline(switch_questions,256);
                        if ((strcmp(switch_questions, "y")==0) or strcmp(switch_questions,
"Y")==0){//continue only if user say yes


                                /*setting the info array, this could be excluded completely if
needed but would
                                 be good to have for at least some generations.
                                 Could limit the number of gen when this is created but for now
                                 it is the same size as the number of gen*/
                                info_per_gen_per_pop = new
saved_information*[burnin_generations+1];
                                tot_fixed_loci = new int[burnin_generations+1];
                                same_fixed_loci = new int[burnin_generations+1];
                                sexratio = new int[burnin_generations+1];

                                //---------These next lines are to create a empty new parental
population lines (and empty new offspring population holder arrays)--
                                subpop = new population[subpopnumber+1];
                                subpop_offspring = new population[subpopnumber+1];
                                for (counter = 1; counter<(subpopnumber+1);counter++) {
                                    //parental
                                    sprintf(inttochar,"%d",counter);
                                    strcpy(subpop[counter].popname, "Population_");
//assigning string values
                                    strncat(subpop[counter].popname, inttochar,5);
//concatenating strings
                                    subpop[counter].popsize = subpopsize;
                                    create_empty_pop(subpop[counter],counter);
                                    //          showpop(subpop[counter]);

                                    //offspring
                                    sprintf(inttochar,"%d",counter);
                                    strcpy(subpop_offspring[counter].popname,
"Offspring_Population_"); //assigning string values
                                    strncat(subpop_offspring[counter].popname,
inttochar,5); //concatenating strings
```

```
                        subpop_offspring[counter].popsize = subpopsize;
                        create_empty_pop(subpop_offspring[counter],counter);
                        //      showpop(subpop_offspring[counter]);
                }

                //----To reproduce for a number of generations, creating the
parental populations, ADD something here
                //record_fraction = 10; //percentage of generations for which the
information is recorded
                //record_fraction = (record_fraction/100)*burnin_generations;


                //REMOVE THIS REPEATER LATER WHEN ALL IS
TESTED
                //      int repeats = 1;
                //      int repcounter;
                //      int samefix, difffix, totfix;
                //      samefix = 0;
                //      difffix = 0;
                //      totfix = 0;
                //      for (repcounter = 1; repcounter<(repeats+1);
repcounter++) {

                 for (counter = 1; counter<(burnin_generations +1); counter++) {
//running for each generation!!!
                        //              cout << "\nGeneration "<< counter << "
";
                        reproduce(counter);
                        //      selection_1();//first type of selection

                        /*      for (counterB = 1;
counterB<(subpopnumber+1);counterB++) {//internal checking things
                         showpop(subpop[counterB]);
                         showpop(subpop_offspring[counterB]);
                         } */
                        record_info_for_gen(counter);//recording some
information for the current generation

                         phenotyping();
                         outputfunction(counter);

                }//END of...reproduce function//for the amount of burnin gen

                //more repstuff
                //cout << "\nRepeat no: "<< repcounter << ", same fixed: "<<
same_fixed_loci[burnin_generations] << " (tot loci "<< haplotype_size<< ")";
                //      if (same_fixed_loci[burnin_generations] == 1) {
                //              samefix = samefix+1;
                //      }
                //      if (tot_fixed_loci[burnin_generations] == 1){
```

```
                                        //          totfix = totfix +1;
                                        //      }
                                        //difffix = totfix - samefix;

                                        //} //repeatruns end

                                        //cout << "\n\n Total Runs :" << repeats;
                                        //cout << "\n Total fixed :" << totfix;
                                        //cout << "\n Total unfixed :" << repeats-totfix;
                                        //cout << "\n same fixed :" << samefix;
                                        //cout << "\n different fixed :" << difffix;
                                        //repeater stuff till here

                                        plot_fixed_allel_freq(); //plotting the allelel freqs after
simulation

                                        //showing the last population
                                //                              for (counterB = 1;
counterB<(subpopnumber+1);counterB++) {
                                //                                      showpop(subpop[counterB]);
                                //                              }


                                }//END of...to continue if the user has cheked the data before simulation


                }//------------------Here ends the simulation of the initial populations before the
experimental cross is done------

                else {//----start to load parentals------------------------
                        cout << "Used when loading parental file. \n";
                }//-------end of loading parentals------------------------

                //DO THE CROSSES HERE IF STILL REQ


        //      phenotyping();

                //----WRITING SOME OUTPUT HERE---
                //bool writeoutput = true;//just a switch
                // if (writeoutput==true) {
                         //who gets reproduced, grandpar = burnin number, F1 = grandpar+1, F2 = F1+1
                        // outputfunction(burnin_generations);//the grandparents or F0.
                 //}


        }//END of...if the files was opened succesfully (xfileopen)

cout << "\n";
```

```
}//ENDING OF MAIN PROGRAM, RMN.
```