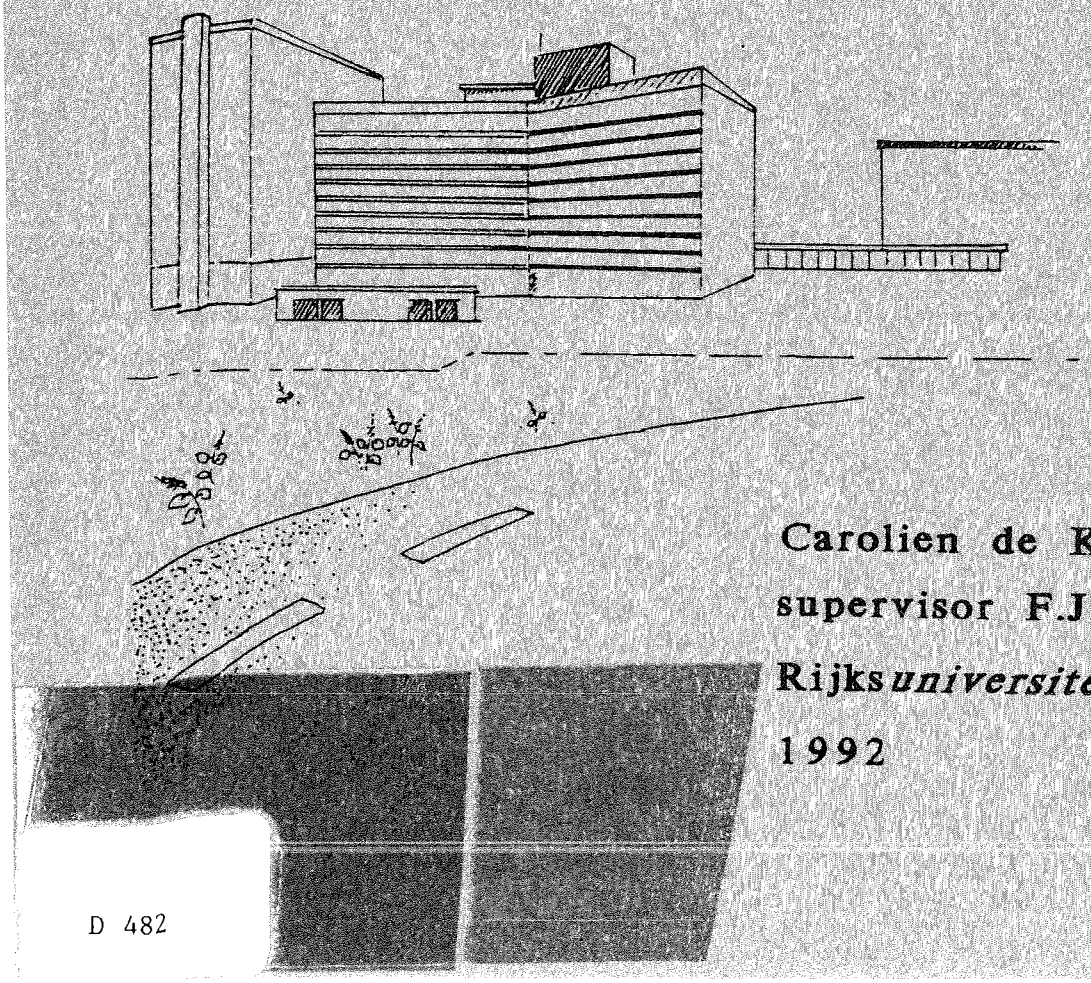
The relative importance of overdominance and partial dominance for the fate of small populations



Carolien de Kovel
supervisor F.J. Weissing
Rijks*universiteit* Groningen

1992

THE RELATIVE IMPORTANCE OF OVERDOMINANCE AND PARTIAL DOMINANCE FOR THE FATE OF SMALL POPULATIONS

Carolien de Kovel
Supervisor Franz Josef Weissing

Department of genetics
Rijksuniversiteit Groningen
december 1992

Abstract

Monte-Carlo computer simulations were used to investigate the relative importance of the partial dominance model and the overdominance model for the fitness and the probability of extinction of small populations.

When 12 loci under partial dominance were combined with 4 loci under overdominance, the effects of partial dominance on the population fitness were on average undetectable within 25 generations for most values of the selection coefficients.

When the fertility of the population was high, partial dominance on 12 out of 16 loci increased the probability of extinction only slightly in the first generations, whereas overdominance on 4 out of 16 loci increased the probability of extinction substantial in all generations. When the fertility was low, partial dominance increased the probability of extinction in the first generations much more than when the fertility was high, but the effects of partial dominance did not exceed the effects of overdominance.

The probability of extinction was higher when the selection against partially dominant alleles was weak (s=0.2), than when the selection was strong (s=1).

It seemed that a long period of weak selection, wether caused by selection against partially dominant alleles or against overdominant alleles, was more harmful to small populations than a short period of rather strong selection.

If overdominance is present on a few loci in a small population, its influence on population fitness and extinction probability may in many cases dominate the influence of partial dominance on many loci.

Introduction

Due to human activities many originally large plant populations have declined suddenly in recent decennia. This happens when for example roads are constructed or land is taken for agriculture. The plants' habitat is fragmented or decreased so there is less room left for the plants and small populations are formed. A small population faces several problems:

-Demographic stochasticity can result in a strongly fluctuating population size. This is because of sampling errors in the number of births and deaths.

-In small populations random processes are more important than in large populations, which are mainly ruled by deterministic processes like selection. Therefore allele frequencies in small populations may change in a different way than in large populations. This is called genetic drift: random changes in allele frequencies that occur due to sampling error, including the loss of alleles (Frankel & Soulé, 1981). Loss of alleles can result in fixation of other alleles.

Small populations suffer a loss of genetic variance that is theoretically 1/2N per generation, where N is the number of individuals in that population. This is 2% per generation for a population of 25 individuals. Relatively rare alleles, that contribute little to the genetic variance, have a high probability of getting lost when the population size is limited (Frankel & Soulé, 1981). These may be alleles for disease resistance that are crucial during special circumstances, but also harmful alleles that are kept in the population by mutation.

-Individuals in small populations get related, so inbreeding occurs. As a consequence there will be less heterozygotes than expected in a large population according to Hardy-Weinberg proportions.

In inbreeding populations multiple heterozygotes and multiple homozygotes are more common than would be expected based on the genotype frequencies (Holsinger, 1988).

Inbreeding as well as fixation of alleles by drift results in a relative shortage of heterozygotes.

Shortage of heterozygotes might be disadvantageous under two selection models: (1) when there is heterozygote advantage (*overdominance*) and (2) when the heterozygote does not have maximal fitness, but one of the homozygotes has low fitness (*partial dominance*).

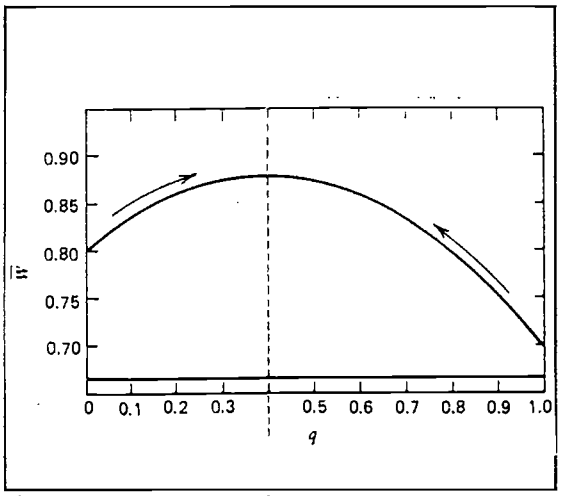


Figure 1, W as a function of allele frequency q on overdominant locus. s1=0.2, s2=0.3. Selection tends to move allele frequencies to a stable equilibrium.(from Li, 1955; Spiess, 1989).

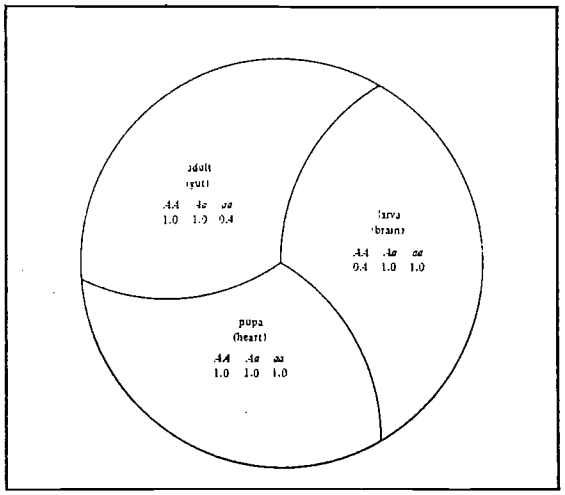


Figure 2. Model for marginal overdominance: Averaged over all stages or tissues the heterozygote is superior to both homozygotes. (Frankel & Soulé, 1981).

The fate of a small population differs dependent on which model is assumed.

The overdominance model

Overdominance means the heterozygote is selectively superior to the homozygotes. In a one locus, two allele situation overdominance can be represented as follows:

genotype	AA	Aa	aa	
fitness	1-s1	1	1-s2	0<s1,s2<1

In these formulae s1 and s2 are selection coefficients against the homozygotes.

The equilibrium frequency in case of overdominance is

$$\hat{p}(a) = \frac{s1}{s1+s2}$$

(Wallace, 1981). At the equilibrium the population fitness is

$$\overline{W} = 1 - \frac{s1*s2}{s1+s2}$$

At this equilibrium the mean fitness of the population, which is determined as

$$\overline{W} = p^2 w_{11} + 2pqw_{12} + q^2 w_{22}$$

is maximum (figure 1).

One form of overdominance are resistance alleles that protect the heterozygote against several diseases, but the

homozygote only against a few. Another form are alleles that protect heterozygotes against herbicides, but which are lethal in homozygote condition.

But overdominance can also occur when a certain allele gives advantage in some processes, whereas the other allele is advantageous in other processes (see figure 2). This is called marginal overdominance (Frankel & Soulé, 1981).

Loss of alleles due to drift as well as shortage of heterozygotes due to inbreeding, which both occur in small populations, will shift the population fitness to a lower value.

The partial dominance model

Partial dominance occurs when one of the homozygotes has the lowest fitness of all genotypes and the heterozygote has intermediate fitness. For a one locus, two allele situation this can be represented as follows:

genotype	AA	Aa	aa	
fitness	1	1-hs	1-s	0<s,h<1

where s denotes the selection coefficient and h the degree of dominance of allele a. A value of h=0.05 seems to be typical for lethal alleles (s=1) whereas a degree of dominance of about 0.3 is often assumed for detrimental alleles (s=0.2) (Gillespie, 1976).

The explanation for the occurrence of polymorphism under partial dominance is that all kinds of alleles that are less functional than the wildtype A-allele, are continually formed by mutation. (All these -physically different- mutant alleles are represented by the letter "a".) The mutation back from a mutant to the original allele A is very rare. The mutant allele is on the one hand rapidly removed by selection, but on the other hand continuously formed by mutation. This way an equilibrium is maintained which is called mutation-selection equilibrium. The frequency of the mutant allele at this equilibrium is

$$\hat{q}(a) \approx \frac{\mu}{hs}$$

In this formula μ is the per locus mutation frequency, s the average selection against the mutant homozygote and h the degree of dominance of the mutant (Wallace, 1981). The equilibrium frequency is very low, because μ is about 10^{-6} (Simmons & Crow, 1977).

The population fitness at this equilibrium is

$\overline{W} \approx 1-2\mu$

In a small population the situation is different. There, selection against the disadvantageous allele will be stronger since the allele will more often appear in homozygous condition. Because the selection in small populations is stronger, the population fitness will be lower than in an infinite population. Since selection against the deleterious allele is strong, it will quickly disappear from the population. If the population survives this period of so-called purging its fitness increases and may temporarily exceed the fitness of an infinite population. So the effects of loci with partial dominance on the fitness of a small population are harmful at first, but beneficial in the long run, unless a mutant allele becomes fixed by genetic drift. Of course this is only the case if there is no deleterious effect of the loss of alleles.

Empirical relevance of the models

In the literature there is much discussion about which model is more important in nature. Both models are able to explain the occurrence of polymorphism, but neither is completely satisfactory. In *Drosophila* it has been shown repeatedly that several recessive lethals are kept in a population (Mukai, 1964; Mukai, Chigusa & Yoshikawa, 1965). It is difficult to show that overdominance occurs at a certain locus, but for example the investigations of Schaal & Levin (1976) indicate it does exist. For an extensive discussion see Lewontin, 1974.

The discussion has important implications for conservation biology. If partial dominance is the main factor in inbreeding depression, a small population that has already gone through a few generations of purging will not suffer much more from genetic problems. If, on the other hand, overdominance is an important factor, a small population will continually need "fresh blood" to revive it, or populations must be kept at a larger population size.

In this study the relative importance of partial dominance and overdominance for the fate of a small population is evaluated. In particular the difference between shortterm and longterm effects is looked into.

The investigations are based on computer simulations. I implicitly assumed that both selection mechanisms occur, but that partial dominance is more frequent.

The model

Structure of the model

A computer programme was used to run simulations of small populations. The simulation model was based on the following assumptions: I considered a population with discrete, non-overlapping generations.

The individuals are diploid and hermaphrodite. There is normal Mendelian segregation in heterozygous parental individuals for pollen and ovule production and all gametes have equal success in producing zygotes. There is no correlation in pollen allele frequencies caused by factors such as pollinator behaviour or pollen morphology. Matings occur at random.

Selection is only possible via viability differences, i.e, there are no fertility differences. The viability of an individual is governed by 16 fitness loci, each with two different possible alleles. There is no linkage between the loci. The fitness effect of a genotype at a certain locus is denoted by W_{AA}, W_{Aa}, W_{aa}. The overall fitness of an individual is the product of its fitness on every locus.

The selection model at a certain locus can be overdominance:

$W_{AA} = 1-s1 \quad W_{Aa} = 1 \quad W_{aa} = 1-s2$

and it can be partial dominance:

$W_{AA} = 1 \quad W_{Aa} = 1-hs \quad W_{aa} = 1-s.$

When on more than one locus overdominance is present, the selection coefficients on all loci with overdominance are equal. The same is true for loci with partial dominance: the selection coefficient and degree of dominance are identical on all loci with partial dominance.

The fitness of the population is the mean fitness of all individuals, including those that did not survive.

For a discussion of common assumptions in (computer) models see Hedrick (1990).

Three models were constructed.

(1) *Infinite population*

This model was made to be able to compare the finite populations to an infinite population.

(2) *Finite population of fixed size N*

In this model there is no demographic stochasticity. It was used to study the course of population fitness in time.

(3) *Finite population of variable size*

In this model demographic stochasticity plays a role. It was used to study extinction.

A comparison of the finite and infinite population was made by the following equation:

$$\delta_N = \frac{\overline{W_\infty} - \overline{W_N}}{\overline{W_\infty}}$$

In this formula $\overline{W_\infty}$ is the fitness of the infinite population and $\overline{W_N}$ is the fitness of a population of size N. The δ_N is called the fitness depression. If the fitness depression is zero, the finite and the infinite population have equal population fitnesses. A positive value of δ_N means the infinite population has higher fitness, whereas a negative value means the infinite population has lower fitness than the small population.

The *infinite population* (model 1) was studied by the recurrence equation (Hartl & Clark, 1989):

$$p' = \frac{p(p*W_{AA} + q*W_{Aa})}{\overline{W}_{locus}}$$

for every locus, after which the fitness per locus was calculated:

$$\overline{W}_{locus} = q^2 * W_{AA} + 2pq * W_{Aa} + p^2 * W_{aa}$$

In these formulae $\overline{W}_{locus}$ is the mean fitness at a certain locus, p is the frequency of allele A at that locus and q is the frequency of allele a at that locus. p′ is the frequency of allele A in the next generation. The population fitness is the product of the fitness at each of the 16 loci.

The *finite populations* were studied by means of Monte-Carlo simulations. Both versions of the model assumed that there is a fixed number of sites where a plant can grow. This total number of sites is called the carrying capacity.

First an initial population of fixed size was constructed with genotypes about Hardy-Weinberg equilibrium at each locus. This was done by randomly assigning the alleles A or a to all loci on both "chromosomes". The probability of assigning

allele a was a given value $q_0(a)$ for that locus. For this initial population, as for all following generations, the population fitness, the mean heterozygosity and the allele frequencies were determined.

Then for the next generation two parents were chosen at random for each of the sites; self fertilisation was possible. From each parent a haplotype was drawn and this haplotype had the possibility to undergo mutation. Mutation from A to a took place with probability $2*10^{-4}$ per haplotype (Simmons & Crow, 1977). Mutation from a to A was considered as too unlikely to be taken into account. The two haplotypes together made up a zygote.

The viability of the zygote W_z was determined on basis of its genotype.

$$W_z = \prod_{locus} W_{A_iA_j}$$

Selection was simulated by removing the zygote from the population with a probability of $1-W_z$. If the zygote survived, i.e. was not removed, it was added to the next generation, if not, a new zygote got its chance.

Model(2) assumed that for each site many seeds are available and new zygotes were drawn until one survived and the site was occupied. Under this model the population size remained fixed at the carrying capacity (corresponding to the total number of sites).

Model(3) assumed that there is a limited number of seeds available per site. The average number of seeds per parent was fixed. So the total number of seeds corresponds to this average number of seeds times the population size. The seeds were not uniformly distributed over the sites but according to a Poisson distribution. This means there may be sites with no seeds as well as sites with many seeds. For each site zygotes could be drawn as described above as long as there were seeds left. If none of the zygotes survived, the site remained empty during the next generation. In that case the carrying capacity was not reached. Not-germinated seeds did not survive till the next generation.

After the whole new generation had been constructed, the old population was replaced by the new one. The fitness of the population, mean heterozygosity etc. were determined.

According to model(1) the fitness of an infinite

population under the same conditions as the small population (same fitness matrix, same initial allele frequencies) was calculated and the finite population was compared to the infinite as described above. The comparison was made in the zygote stage, that is, before selection had taken place.

To evaluate the influence of demographic stochasticity in model(3), the expected population size without selection was calculated. This was done by the following calculations:

If s is the average number of seeds per parent and N is the population size of the parent population, the total number of seeds available is $N*s$. The mean number of seeds per site (μ) is given by

$$\mu = N*s/K$$

where K is the total number of sites (carrying capacity). According to the Poisson distribution, the probability of x seeds at a certain site is

$$p(x) = e^{-\mu} \cdot \frac{\mu^x}{x!}$$

The probability of at least one seed at a certain site is

$$1-p(0) = 1-e^{-\mu \cdot \frac{\mu^0}{0!}} = 1-e^{-\frac{Ns}{K}}$$

If multiplied with K, the total number of sites, this equation gives the population size in the next generation if all loci are selectively neutral. Iteration provides the development of the population size without selection. These iterations lead to a stable equilibrium > 0 for the population size, if $s > 1$.

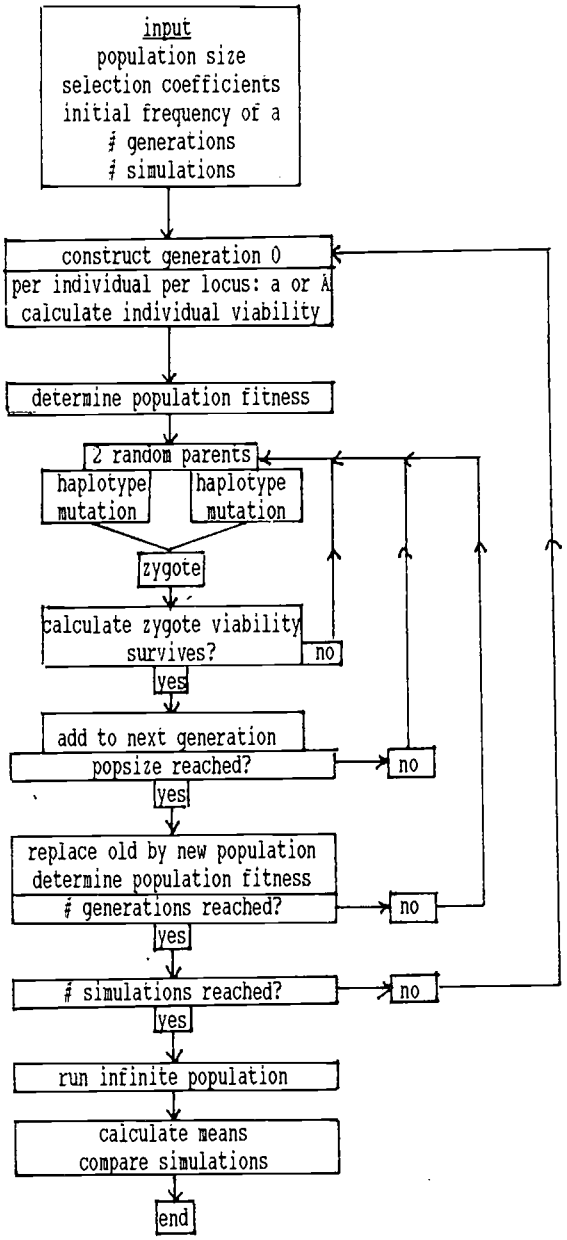
The simulation programmes were written in Turbo Pascal 5.5. The set-up of the programmes is represented by the flowcharts figure 3. The source code of the programmes can be found in the appendix, nr 1.

Results

Overdominance

The effects of overdominance on population fitness were studied by running simulations with overdominance on all 16 loci. The simulations started at about equilibrium frequencies

Flowchart Program Inbreed1.pas

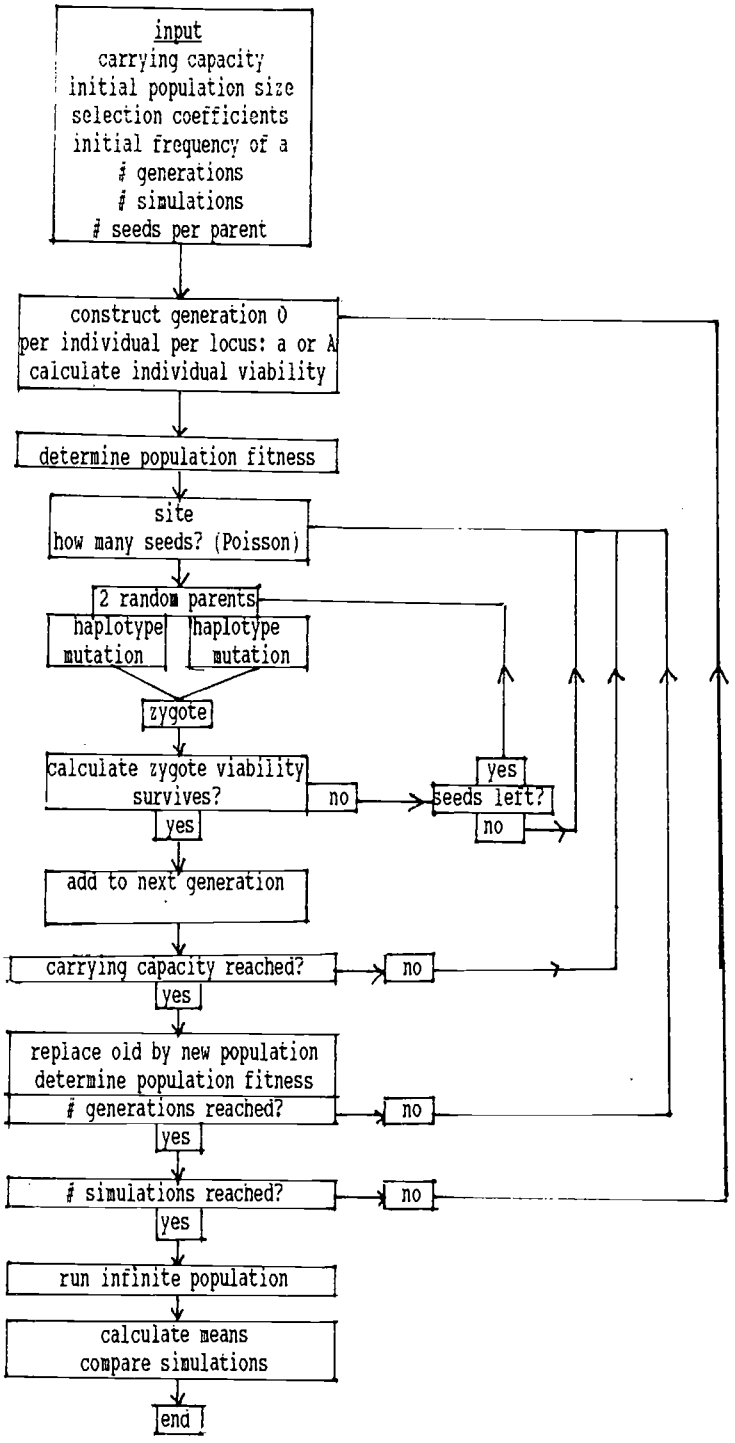


Possible output:

-Population viability finite population
  mean over simulations + standard deviation
-Population viability infinite population
-Relative viability
  mean + standard deviation
-Frequency of allele a
  mean
-Coefficient of gene differentiation between simulations: G_{st}
-Genetic identity between first 3 simulations (M.Nei in Spiess, 1984)
-Fixations
-populationsize before selection

figure 3a. Flowchart of programme INBREED1.PAS (model 2).

Flowchart Program inbreed3.pas



possible output
-as in inbreed1.pas
-populationsize
-moment of extinction

Figure 3b. Flow chart of programme INBREED3.PAS (model 3).

for the alleles. When overdominance was symmetric, the small population had a steadily increasing fitness depression (see figure 4). This means the small population had an increasing fitness disadvantage relative to the infinite population. The maximum possible fitness depression of $(1-s1)^{16}$ was not reached within 150 generations, which means not all loci were fixed yet at that moment. When overdominance was very asymmetric the fitness depression increased steeply in the first generations and reached a plateau (see figure 5). The increase depended on population size, selection coefficients and skewness of the selection. In some cases of asymmetric overdominance a top was reached and after that top the disadvantage diminished; this probably means the population was purged of the least fit homozygotes (not shown).

There was not very much variation in the general pattern, especially when the overdominance was (nearly) symmetric. This can be seen in figure 6.

The fitness depression of the finite population increased faster when the population was small (see table 1). This is because inbreeding is more severe and drift is more important if the population size is small. In table 2 the mean fitness and fitness depression of a population of size 25 after 150 generations are shown for several selection coefficients. It can be seen that the selection against the fittest homozygote was more important for the mean fitness of the finite population than the selection against the less fit homozygote. This was not the case for the fitness depression.

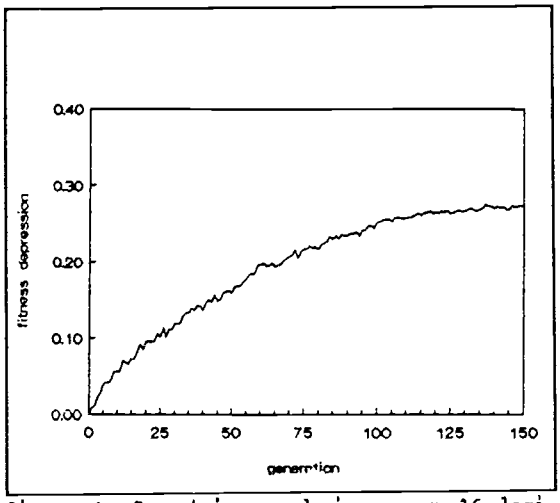


figure 4. Symmetric overdominance on 16 loci. Population size = 25, s1=s2=0.05. Mean of 45 simulations.

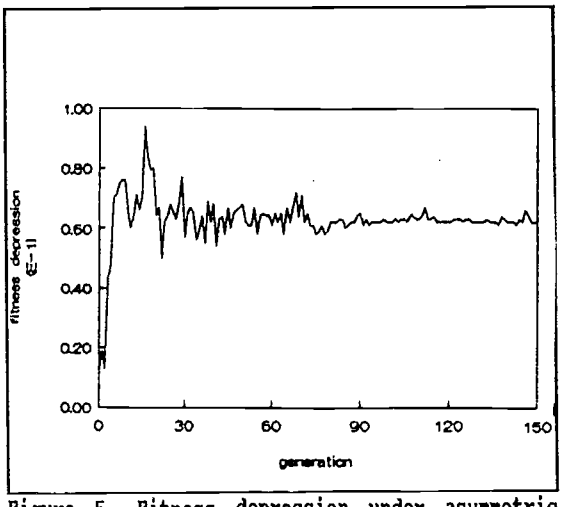


Figure 5. Fitness depression under asymmetric overdominance on 16 loci. Population size = 25. s1=0.6, s2=0.05 p_0=0.1. Mean of 10 simulations.

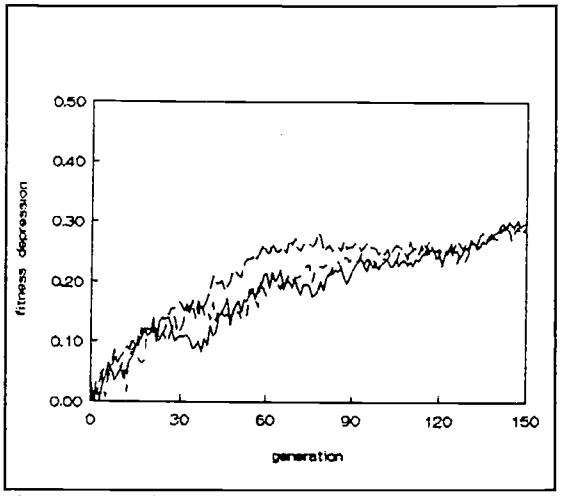


Figure 6. Fitness depression in 3 different simulations with 16 loci under overdominance. Population size = 25. s1=s2=0.05.

Table 1. Fitness depression after 100 generations in populations of different size. Symmetric overdominance on 16 loci. $W_{AA}/W_{Aa}/W_{aa}$ = 0.95/1/0.95. Mean of 50 simulations.

POPULATION SIZE	fitness depression δ_N
N=100	0.05
N=50	0.15
N=25	0.29
N=10	0.33

Table 2. Fitness and fitness depression of a population of size 25 after 150 generations under different selection regimes. Overdominance on 16 loci. Initial allele frequencies are equilibrium frequencies. Mean of 45 simulations.

SELECTION AA/Aa/aa	fitness	fitness depress. δ_N
0.95/1/0.95	0.47	0.29
0.99/1/0.99	0.86	0.07
0.99/1/0.95	0.81	0.06
0.95/1/0.40	0.44	0.06

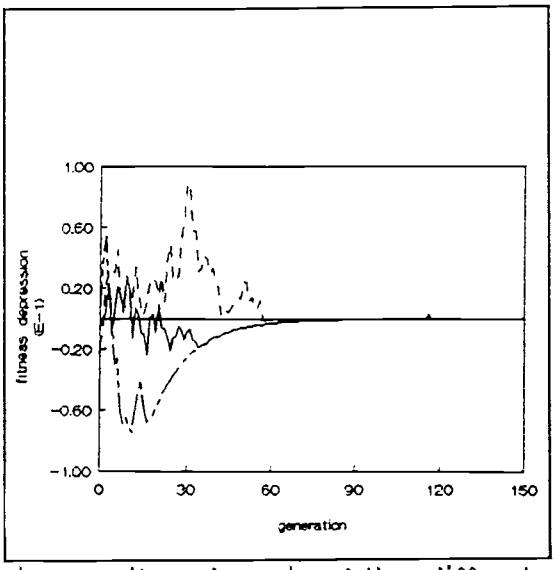


Figure 7. Fitness depression of three different simulations under partial dominance on all 16 loci. q_0=0.1 s=0.2, h=0.35, population size = 25.

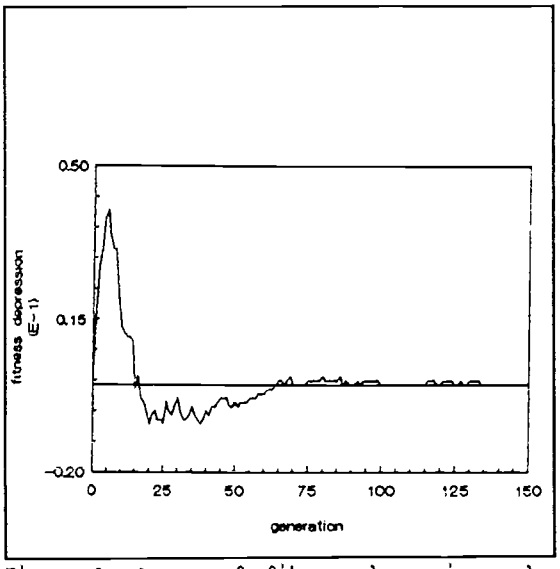


Figure 8. Course of fitness depression under partial dominance on 16 loci. q_0=0.1, s=0.2, h=0.35, population size is 25.

When compared to pure drift effects (no selection) fixation was significantly delayed by symmetric overdominance of 5% ($p<0.05$).

Partial dominance

When there was partial dominance at all 16 loci, the small populations had on average a lower population fitness than the infinite population during the first generations. There was however much variation between the simulations and in some of the simulations the small populations even had a higher fitness in the first generations than the infinite population (see figure 7). In figure 8 the typical course of the fitness depression can be seen. In this picture the mean fitness depression of 10 simulations is presented.

The magnitude of the maximum fitness depression of the small populations depended on population size, selection and initial frequency of the mutant allele. In most simulations the initial frequency of allele a was 10%. This is higher than the theoretical value based on mutation-selection balance, which is about 10^{-4} (Simmons & Crow, 1977; Wallace 1981). I used this value because otherwise mutant allele a would hardly have a chance to appear in the small population.

The moment the fitness depression of the small population was maximum and the number of generations till the depression had been reduced to zero depended mainly on the selection coefficients.

Table 3. Largest and smallest δ_N and the generations they occur in for several population sizes and selection coefficients. For all simulations q_0 = 0.1. The values are calculated as the running mean of 50 simulations over 10 generations.

population size	selection	largest δ_N	generation	smallest δ_N	generation
25	s=0.1 h=0.05	0.027	34	0.004	92
25	s=0.4 h=0.05	0.027	16	-0.010	47
50	s=0.1 h=0.05	0.014	28	-0.004	113
50	s=0.4 h=0.05	0.018	15	-0.007	57
50	s=0.4 h=0.45	0.023	14	-0.002	48
50	s=0.8 h=0.05	0.019	14	-0.006	42
100	s=0.1 h=0.05	0.008	32	-0.002	121
100	s=0.4 h=0.05	0.012	17	-0.004	73

Maximum fitness depression δ_N, is largest in small populations with high selection coefficients. For recessive lethals (s=1) with a degree of dominance (h) of 0.05 maximum depression in a population of size 10 was on average 0.21. This maximum fitness depression occurred in the 4th generation. The highest fitness depression found under these conditions was 0.55. The same coefficients in a population of size 25 produced on average a maximum depression of 0.16 (mean of 20 simulations). Again this maximum occurred on average in the 4th generation. The depression had been reduced to zero after 11 generations. When the selection was much weaker (s=0.2 and h=0.05) the mean maximum depression in a population of size 25 was 0.07 in the 20th generation. With these coefficients it took on average 42 generations for the depression to diminish to zero.

I determined the values of maximum and minimum δ_N and the generation they appeared in by plotting the running mean of

Table 4. Largest and smallest δ_N and the generations they appear in for several values of q_0. selection coefficients: s=0.4 h=0.05. Running means over 10 generations of 50 simulations.

population size	q_0	largest δ_N	generation	smallest δ_N	generation
50	0.1	0.018	15	-0.007	57
50	0.3	0.047	14	-0.008	66
50	0.9	0.654	14	0.345	68

the fitness depression over 10 generations. The outcome of this is presented in table 3. The procedure of calculating the running mean of a number of simulations however flattens out the extremes and shifts them to the right. Therefor δ_N does not attain its maximum in this table before generation 14 and extremes are lower than mentioned above.

The influence of different initial allele frequencies can be seen in table 4 (also determined by using running mean). Because the high frequencies are in normal circumstances very unlikely, they will not be discussed further.

When s<0.1 the mutant allele has a significant probability to become fixed under the conditions of the model. As a consequence in some populations purging never gets complete.

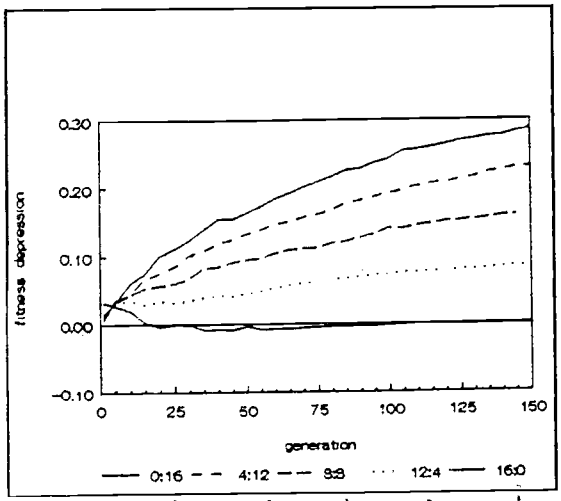
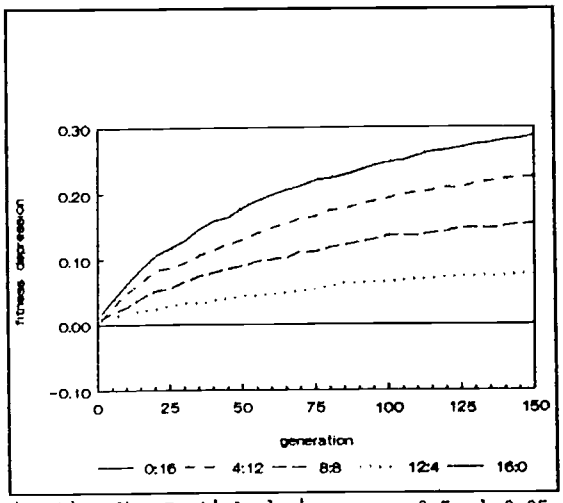
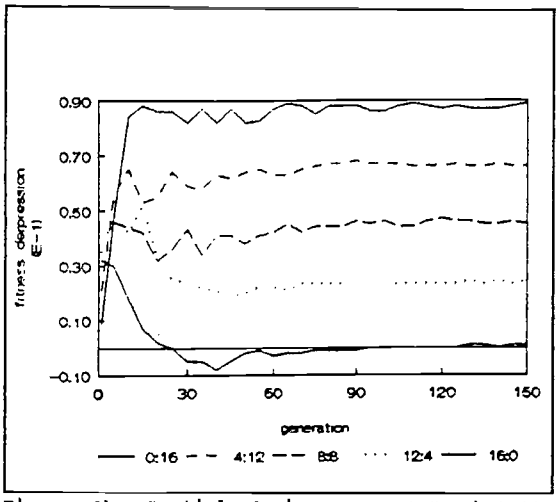


Figure 9a. Fitness depression under various combinations of partial dominance (left) respectively no selection (right) and overdominance. Population size = 25.



(continued). Partial dominance: s=0.5, h=0.05. Overdominance: s1=s2=0.05.



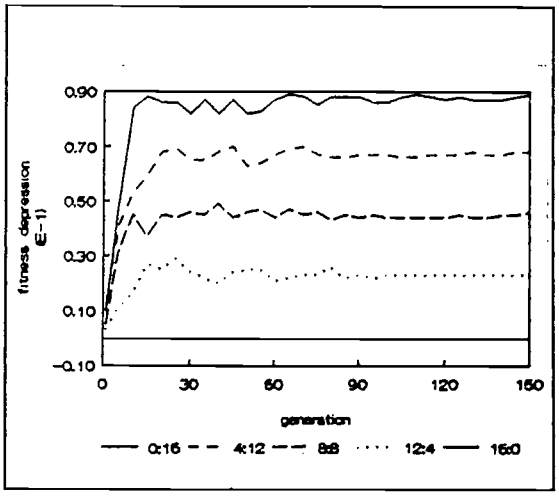


Figure 9b. Partial dominance: s=0.5, h=0.11. Overdominance: s1=s*h, s2=s, $p_{0=}0.1$.

Course of population fitness under combination of overdominance and partial dominance

When partial dominance and overdominance were combined, there were three stages: first the stage in which loci under partial dominance have the biggest influence because of mutant homozygotes that cause lower fitness. Second there was in some cases a stage of combined influence of purging, which raises the fitness of the small population, and overdominance, which lowers the fitness. Third there was the stage when the influence of partial dominance was undetectable and there was only the influence of overdominant loci.

In figures 9 the influence of the loci with partial dominance is shown. In these figures populations with partial dominance on n loci and overdominance on 16-n loci are compared to populations with n neutral loci and 16-n overdominant loci.(To show the average effects, the figures present the mean of 45 simulations.)

When 4 of the 16 loci were under symmetric overdominance (s1=s2=0.05) and the remaining 12 loci were under partial dominance with s=1 (lethal) and h=0.05, the influence of the lethal mutants disappeared in less then 10 generations. This was the case even though the initial frequency of the lethal mutants was as high as 10%. The negative influence of loci with partial dominance had for most reasonable values for h and s (Simmons & Crow, 1977) disappeared in about 25 generations. Of course this only holds true if no mutant allele had

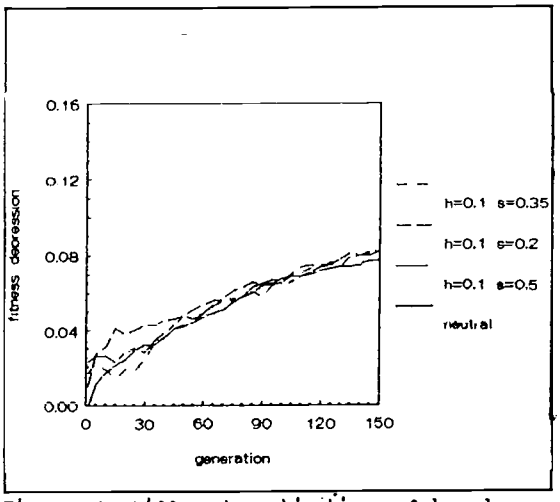


Figure 10. Different combinations of h and s on 12 loci under partial dominance. Overdominance on 4 loci with s1=s2=0.05. Pairs with equal value of h*s are presented. Population size = 25.

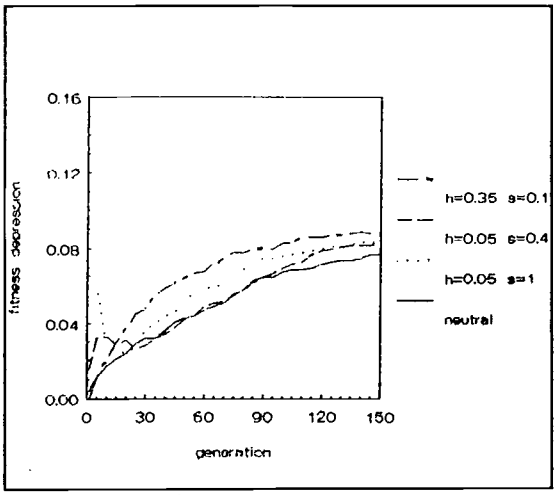


Figure 10 (continued).

become fixed on one of the loci. Fixation of the mutant allele though appeared to be rare under the conditions of the model.

In infinite populations the per generation change in fitness depends on the value of the product h*s, the selection against the heterozygote (Wallace, 1981). In figure 10 it can be seen that this is not the case for small populations. In this figure combinations of h and s with the same product are depicted. In figure 11 it can be seen that the larger s is, the shorter is the influence of partial dominance. The influence of h is less clear, but there seems to be the same tendency.

The relative importance of overdominance for extinction chances of small populations

In model 3 (variable population size) simulations were run with 12 partially dominant loci and 4 overdominant loci. This programme was very sensitive to changes in average number of seeds per parent. When this was low all populations went extinct due to demographic stochasticity even when there was no selection.

Selection results in decreasing population size, but demographic stochasticity delivers the final blow. This was seen when populations went extinct, even though the population fitness was constant for many generations. Often the fitness was rather high or even unity at the moment of extinction. This means that the history of the population and the random

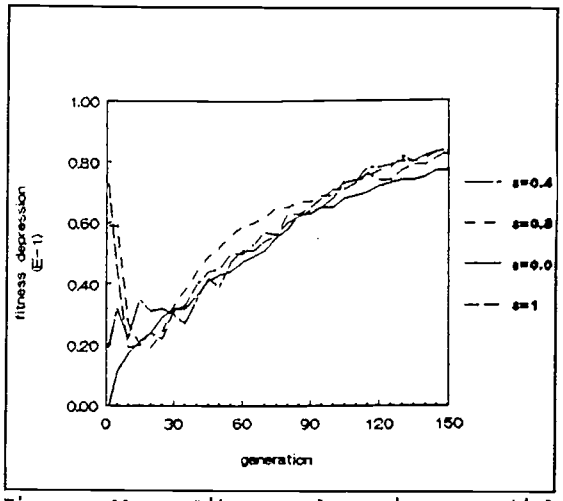


Figure 11a. Fitness depression. Partial dominance on 12 loci with different values for s, h=0.1. Overdominanc on 4 loci: s1=s2=0.05. Population size = 25.

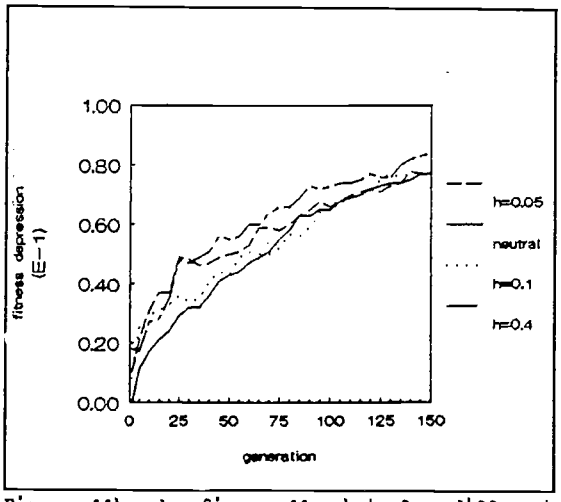


Figure 11b. As figure 11a but for different values for h, s=0.2.

processes are important factors for the fate of the small population. It was also seen in simulations with overdominance only, that there can be a positive feedback between population size and population fitness: smaller populations have a higher degree of inbreeding, so they have a faster decrease in population fitness, fewer individuals survive and the circle is closed. But again random processes are important: under the same starting conditions some populations did end up in this circle and others did not, some could break free from it and others could not.

The initial population size was less important than the possible population size, the carrying capacity for the probability of extinction. Frankel and Soulé (1981) also concluded that a single generation bottleneck is not as harmful as a long period of small population size.

In table 5a and 5b it is shown that lethal and semi-lethal alleles did not significantly increase a small population's chance to go extinct, provided enough seed was produced and the carrying capacity was not too small. Overdominance on a few loci however greatly increased the probability of extinction. Strange enough this did not happen mainly after many generations when most heterozygotes had disappeared, but also early in the simulations.

These findings suggest that overall population fitness alone is a poor indicator of the population's capability to survive.

Most important factor of overdominant loci on the

probability of extinction and population fitness was the fitness of fittest homozygote. Lowering the fitness of the less fit homozygote did not increase the extinction probability much (table 5).

When the fertility was low (table 6a,b) or the carrying capacity was small (<20, not shown), the influence of partially dominant alleles increased, but it dominated in none of the simulations.

When the value of s on the partially dominant loci was low, more extinctions occurred than when the value of s was high (table 5). This seemingly paradoxical result was not caused by fixation of mutant alleles, for in none of the simulations that went extinct that happened, whereas mutants did become fixed in some simulations that did not go extinct.

Under the overdominant model the distribution of the extinctions over the generations did not differ significantly from uniform distribution when conditions were not very severe. Under the partial dominant model it does (Chi-square, $p<0.05$). When there was only partial dominance on 12 loci and no selection on the other 4 loci, more than half of the

Table 5. Number of populations going extinct in less than 150 generations in 45 simulations.
vertical:12 part. dominant loci, horizontal:4 overdominant loci: $W_{AA}/W_{Aa}/W_{aa}$
N(0)= 25, (a):carrying capacity (K) =25, (b): K=50, average number of seeds = 1.5. (-): no results.

SELECTION	neutral	0.95/1 /0.95	0.99/1/ 0.99	0.9/1/ 0.7	0.4/1/ 0.95
neutral	1	43	4	45	43
s=1 h=0.05	1	41	-	45	44
s=0.8 h=0.05	1	39	1	45	39
s=0.2 h=0.05	9	39	-	-	43
s=0.2 h=0.35	11	41	3	-	42
s=0.1 h=0.35	10	45	-	-	44

(a)

SELECTI-ON	neutral	0.95/1 /0.95	0.99/1/ 0.99	0.9/1/ 0.7	0.4/1/ 0.95
neutral	0	13	0	45	18
s=1 h=0.05	0	17	-	45	15
s=0.8 h=0.05	0	19	0	45	15
s=0.2 h=0.05	0	-	-	-	27
s=0.2 h=0.35	1	20	1	-	24
s=0.1 h=0.35	0	22	-	-	22

(b)

extinctions occurred in the first 40 generations. When conditions were poor extinctions under overdominance model also occurred mainly in the first generations.

The expected population size at equilibrium without selection is presented in table 7. This expected population size does not depend on population size in generation 0. The moment the equilibrium is reached, does depend on the initial population size. This equilibrium population size is proportional to the total number of sites, K, as might be expected from the original equation.

If selection is important in the process of extinction, one might expect populations that go extinct to have lower fitness than populations that survive. This is sometimes found in laboratory experiments with populations started with closely related individuals, though not always (Wright, 1977). Anyway it was not the case in the simulations. Some populations recovered from size 1 when population fitness was constantly 0.815, while others under the same circumstances dropped from size 20 to extinction in a few generations. So as for an individual low population fitness just increases the chance of extinction. It is however more complicated than for a single individual, because population size, carrying capacity and mode of dispersion are also important factors.

In figure 12 viabilities of populations that survive and

that go extinct are presented.

Table 6. Number of populations going extinct in less than 150 generations in 45 simulations. vertical: 12 part. dominant loci, horizontal: 4 overdominant loci.
N(0)= 25, carrying capacity (K) =50, (a) average number of seeds = 1.25. (b) average number of seeds =1.20.

SELECTION	neutral	s1=s2=0.01
neutral	10	21
s=1 h=0.05	14	26
s=0.2 h=0.35	19	28

(a)

SELECTION	neutral	s1=s2=0.01
neutral	21	37
s=1 h=0.05	31	39
s=0.2 h=0.35	36	44

(b)

Table 7. Expected population size when no selection is present. Seeds = average number of seeds per parent, K = total number of sites (carrying capacity).

N(equilibrium)	K = 25	K = 50
seeds = 2.00	20	40
seeds = 1.50	15	29
seeds = 1.25	9	19
seeds = 1.20	8	16
seeds = 1.00	0	0

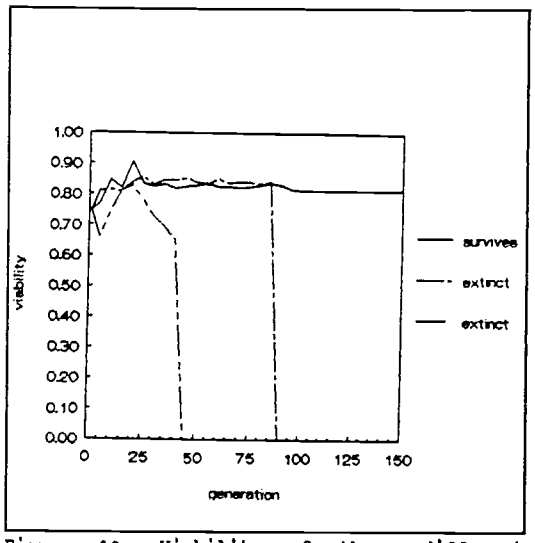


Figure 12. Viability of three different populations under the same conditions: Carrying capacity = 50, average number of seeds = 1.5, 12 loci partial dominance: s=0.2 h=0.35, 4 loci overdominance: s1=s2=0.05.

Conclusions and discussion

Relevance of the assumptions

The findings of this study are relevant if the actual conditions of a plant population resemble the conditions of the model. In the model the population has discrete, non-overlapping generations. This will in most cases mean the plants are annuals. If the generations are not discrete, genetic drift will be less and inbreeding is retarded. The deleterious effects of partially dominant alleles will last longer, but will be less strong. In the simulations the longer lasting effects of relatively weak partial dominance seem to have more impact. However when the population is larger, the effects of partial dominance diminish (table 5), so probably a population with overlapping generations will be in a less precarious position than a population with discrete generations.

A second property of the model population is that it is hermaphroditic and mates at random. Quite a number of plant species are not hermaphroditic and these will have a smaller effective population size (Gregorius, 1991). Random mating is often not the case. Pollinator behaviour, for example, can cause deviations from random mating. Pollinators, such as bees, carry lumps of pollen from one plant, so the next plant

they visit has a high probability of receiving pollen grains that are all related. Furthermore the insects go from one plant to the next and will often pollinate a plant with pollen from the individual standing next to it. This will increase the inbreeding in the population and so accelerate the processes described in this paper (Willson, 1984; Hedrick, 1990).

The model assumes all individuals have equal fertility and the fertility is constant in time. It is however very likely that fertility has a genetic component, as is demonstrated for *Drosophila* (Simmons & Crow, 1977). Loci that govern fertility will probably suffer in a similar way from inbreeding and drift as viability loci do. The population fertility may be lowered by these processes and because of this the population may decrease. In addition, a small population is often less visited by insects (Jennersten, 1988) and because of that the population fertility may be lowered. To compensate for this the plants may self-fertilize more, which leads to a higher degree of inbreeding (Charlesworth and Charlesworth, 1987; Hedrick, 1990). This means that real populations will have a higher probability of going extinct than the model population.

The model is important when most deaths in the population are the result of selection. In many annual species though lots of seeds are produced and the major part of the mortality is due to random, environmental causes, such as grazing, trampling and bad luck in finding a habitable spot. When most of the zygotes die anyway, a few more or less by selective causes do not make much of a difference. However, as discussed above, the fertility of a small population may be severely diminished and in that case selective deaths become important.

The model assumes that partial dominance and overdominance are the only forms of selection that are important. There are many other forms of selection that may be important as well. Other selection models are e.g. *frequency dependent selection*: the least frequent genotype has highest fitness, and *density dependent selection*: genotypes that have high fitness when density is low, have low fitness when density is high. How important these modes of selection are for the fate of small populations, remains to be investigated. Furthermore there may be all kinds of relations between the loci and that of course may also influence the fate of the population.

In the simulations the selection against the different genotypes did not change in time. In a real population the selection may well fluctuate in time (See Hedrick, 1986). Because of changes of the environment, the fittest genotype of

this generation does not have to be the fittest genotype in the next generation. Alleles that are neutral or slightly deleterious under normal circumstances, might be advantageous under specific conditions, such as a sudden outburst of illness or parasites (O'Brien *et al.*, 1985). A large population retains these alleles in small frequency and is able to respond to such catastrophes. In a small population these alleles may get lost rapidly. As a consequence of the loss of these alleles small population may go extinct.

Many other types of environmental changes are possible. The roles partial dominance and overdominance play depend on the character of these changes.

The importance of the strength of selection

In the present study it was found that overdominance and partial dominance with low selection coefficient increase the probability of extinction of a small population more than partial dominance with high selection coefficient. The effects of very disadvantageous recessive alleles do not last long. It seems that a long period of slightly lowered fitness is more disastrous for a small population than a short period of very low fitness. This means population fitness alone is not an accurate indicator of the population's chances of extinction. The history of the population is also important.

The relative effect of overdominance and partial dominance

In a theoretical study, Lande and Schemske (1985) claim that overdominance is of little importance for inbreeding populations. Based on a quantitative genetic model of stabilizing selection on polygenic traits they conclude that most effects of inbreeding observed in nature can be explained by assuming partial dominance.

In a review of the theoretical and empirical literature Charlesworth and Charlesworth (1987) reach a similar conclusion. They point out that detrimental recessives have been found in many populations. These detrimentals may cause a substantial inbreeding depression. In addition the dominance variance with respect to fitness, found in populations, is not large enough in relation to additive genetic variance to be generated by overdominance alone.

The results of the present study suggest that stating that partial dominance occurs in the majority of cases, is not sufficient to reject the possibility that overdominance plays

an important role.

In fact it was found in this study that overdominance on a few loci can influence the fate of a small population very much. The effects on population fitness of deleterious alleles on loci under partial dominance disappear on average within 25 generations. When the fertility is high enough, partial dominance hardly increases the probability of extinction, whereas overdominance on a few loci does.

Charlesworth & Charlesworth (1987) suggest that the effects of overdominance are smaller than claimed in many studies on the topic. In most models on overdominance, overdominance is assumed to be symmetric. This will very seldom be the case in natural situations. Charlesworth and Charlesworth show that in a model with a significant amount of selfing, asymmetric overdominance will lead to fixation of the allele with the higher fitness in homozygous condition, because no equilibrium can be maintained (see Kimura and Ohta, jaar). But also in models in which random mating occurs, fixation happens much more frequently when overdominance is asymmetric. This is the case because the equilibrium frequency of the allele which has the higher fitness in homozygous condition, is close to fixation. Random processes will much sooner lead to fixation of the allele than in the case of symmetric overdominance.

The significance of fixation in the context of overdominance however, is judged differently by Charlesworth and Charlesworth than it is in the present study.

It was found in this study that in the case of asymmetric overdominance, the fitness of the fitter homozygote is more important for the survival of a population than the fitness of the less fit homozygote. Both symmetric and asymmetric overdominance lead to an increase of the probability of extinction and of the fitness depression.

Charlesworth and Charlesworth (and others) use a different measure for the effects of inbreeding than the present study. They compare inbred and outcrossed progeny in a partially selfing population with respect to the inbreeding depression which was defined by

$$\frac{w_o - w_i}{w_o}$$

where w_o is the fitness of offspring produced by outcrossing and w_i is the fitness of offspring produced by selfing in the same population.

Using this measure, they find that overdominance only

contributes to inbreeding depression if a polymorphism is maintained. If alleles become fixed, the fitnesses of the inbred and outcrossed progeny are equal and the inbreeding depression is zero.

Charlesworth and Charlesworth focus on the effects of inbreeding from one generation to the next. The present study is concerned with long-term effects of inbreeding on populations and therefore compares the fitness of the total progeny of a small random mating population to the fitness of the progeny of a totally outcrossing, infinite population. Already after a few generations, the effects of partial dominance become undetectable. The comparison made in this study is more relevant in order to judge the capability of a small population to survive.

One should however notice the large variance of the results (e.g. figure 6). The findings discussed here are based on averages. Because there was variation between different simulations under partially dominant selection, this probably is also the case between different populations that are subject to partially dominant selection. This means that in certain small populations partial dominance may be a more important factor for fitness depression than stated here, even if some loci in the population are under overdominant selection.

Summarizing, this present study does not proof that overdominance exists or that it is important for the fate of inbreeding populations, but it suggests that if overdominance exists on a few loci, its effects on population fitness and probability of extinction may in many cases dominate the effects of partial dominance.

Acknowledgements

I wish to thank Franjo Weissing for all his help during this project and the Department of genetics for the opportunity to learn about all sorts of aspects of scientific research.

References

Charlesworth, D. and Charlesworth, B. 1987. Inbreeding depression and its evolutionary consequences, *Ann. Rev. Ecol. Syst.*18: 237-268.

Frankel, O.H. and Soulé, M.E. 1981. *Conservation and evolution*, pp. 30-77, Cambridge University Press, Cambridge.

Gillespie, J.H. 1976. A general model to account for enzyme variation in natural populations. II.Characterization of the fitness functions, *Amer. Natur.* **110**: 809_821.

Gregorius, H.-R. 1989. On the concept of effective number, *Theor. Pop. Biol.* **40**: 269-283.

Hartl, D.L. & Clark, A.G. 1989. *Principles of population genetics*, pp. 151-153, Sinauer Associates Inc. Publishers, Sunderland.

Hedrick, P.W. 1986. Genetic polymorphism in heterogeneous environments: a decade later, *Ann. Rev. Ecol. Syst.* **17**: 535-566.

Hedrick, P.W. 1990. in *Population biology: ecological and evolutionary viewpoints* (Wohrman, K. and Jain, S. eds), pp. 83-114, Springer Verlag, New York.

Holsinger, K.E. 1988. Inbreeding depression doesn't matter: the genetic basis of mating-system evolution, *Evolution* **42(6)**: 1235-1244.

Jennersten, O. 1988. Pollination in *Dianthus deltoides* (Caryophyllaceae): Effects of habitat fragmentation on visitation and seed set, *Conservation Biology* **2(4)**: 359-366.

Kimura, M. & Ohta,T. 1971. *Theoretical topics in population genetics*. Princeton University Press, Princeton.

Lande, R. and Schemske, D.W. 1985. The evolution of self-fertilization and inbreeding depression in plants. I. Genetic models, *Evolution* **39(1)**: 24-40.

Lewontin, R.C. 1974. *The genetic basis of evolutionary change*. Columbia University Press, New York.

Mukai, T. 1964. Th genetic structure of natural populations of *Drosophila* melanogaster. I. spontaneous mutation rate of polygnes controlling viability. *Genetics* **50**: 1-19.

Mukai, T., Chigusa, S. and Yoshikawa, I. 1965. The genetic structure of naturel populations of *Drosophila melanogaster*. III. Dominance effect of spontaneous mutant polygenes controlling viability in heterozygous genetic backgrounds, *Genetics* **52**: 493-501.

O'Brien, S.J. *et al*. 1985. Genetic basis for species vulnerability in the cheetah, *Science* **227**: 1428-1434.

Schaal, B.A. and Levin, D.A. 1976. The demographic genetics of *Liatris cylindracea* Michx. (Compositae), *Amer. Natur.* **110**: 191-206.

Simmons, M.J. and Crow, J.F. 1977. Mutations effecting fitness in *Drosophila* populations, *Ann. Rev. Genet.* **11**: 49-78.

Spiess, E.B. 1989. *Genes in populations*. John Wiley and sons, New York.

Willson, M.F. 1984. in *Perspectives on plant population ecology* (Dirzo, R. & Sarukhan, J. eds), Sinauer Associates Inc. Publishers, Sunderland.

APPENDIX

Appendix 1: Source code of computer programme Inbreed1.pas (model 2 + 1).

```
0001:
0002: PROGRAM Inbreeding1;
0003:
0004:    {*************************************************************************
0005:     * This program simulates the effects of inbreeding and selection in    *
0006:     * small, non-selfing populations on population-fitness and             *
0007:     * genotype-frequencies.                                                *
0008:     * There are 16 uncoupled loci. Each locus has two possible alleles :   *
0009:     * a and A.                                                             *
0010:     * The fitness of an individual is the product of its fitness at        *
0011:     * every locus. Mutations are allowed for. The generations are          *
0012:     * seperated.                                                           *
0013:     * Populationsize, allele-frequencies and selection may be modified in  *
0014:     * the program.                                                         *
0015:     * -Carolien de Kovel-                                                  *
0016:     *************************************************************************}
0017:
0018: uses Crt;
0019:
0020: CONST
0021:    popsize    = 100;
0022:    generation = 150;                                    {15 at least}
0023:    simulation = 15;                                      {3 at least}
0024:    MeanFile   = True;                                {write means to file ?}
0025:    DepFile    = False;                             {write depression to slide ?}
0026:    Filename1:STRING = 'file1';
0027:    Filename2:STRING = 'file2';
0028:
0029: TYPE
0030:     Haplotype  = word;
0031:     Diplotype  = ARRAY [0..1] OF Haplotype;
0032:     FreqTable  = ARRAY [1..16, 0..2] OF integer;
0033:     Individual = RECORD
0034:                    Geno     : Diplotype;
0035:                    Viability: real;
0036:                    Fertility: real;
0037:                    END;
0038:     Population = ARRAY [1..Popsize] OF Individual;
0039:     AllelTable = ARRAY [1..16] OF single;
0040:     Matrix     = ARRAY [1..16, 0..2] OF single;
0041:     str70      = STRING[70];
0042:     PTable     = ARRAY [1..simulation, 1..16] OF single;
0043:     TotArray   = ARRAY [0..generation, 1..simulation] OF  single;
0044:     MeanArray  = ARRAY [0..generation] OF single;
0045:
0046: VAR
0047:     j,k,gen,sim,x,n          :integer;
0048:     l                        :byte;
0049:     Hetero, Gst              :single;
0050:     Viability                :single;
0051:     s,h                      :single;
0052:     Survive                  :boolean;
0053:     Haplo                    :Haplotype;
0054:     Diplo                    :Diplotype;
0055:     VMat                     :Matrix ;
0056:     GenoFreq,ActGenoFreq     :FreqTable;
0057:     Indi, parent             :Individual;
0058:     Pop, Pop2                :Population;
0059:     p0,pa,pa2                :AllelTable;
0060:     Penc                     :PTable;     {mean pA in last generation}
0061:     Meanstr,DepStr           :str70;
```

```
0062:     PopStr                    :str70:
0063:     ActViab,Ppd,Ph,PopViab    :TotArray:
0064:     ActMean,PopMean           :MeanArray:
0065:     PopS,ActS,DS              :MeanArray:
0066:     Wtot,D,Dw                 :MeanArray:
0067:     PpdMean,PhMean            :MeanArray:
0068:     outfile                   :text:
0069:     ActPopsize                :integer:
0070:     I                         :ARRAY[1..3] OF single:
0071:
0072:                       {**************************}
0073:
0074: PROCEDURE WriteWord (w : Word; wstr : STRING);
0075:     {Write binary code of w}
0076:
0077: BEGIN
0078:    Write(wstr,' =   ');
0079:    FOR j := 15 DOWNTO 0 DO
0080:     IF Odd(w SHR j) THEN Write ('1 ')
0081:                     ELSE write('0 ');
0082: END;      {of WriteWord}
0083:
0084:
0085: PROCEDURE OpenFile;
0086:     {This procedure opens files for various simulations in slide
0087:      or TP, not both!}
0088:
0089: BEGIN
0090:    IF DepFile AND MeanFile THEN Halt;
0091:           IF x = 1 THEN BEGIN
0092:              n:=12;
0093:              s:=0.1;
0094:              h:=0.05;
0095:              Filename2 := 'C:\SW\carodata SDH_01.dat';
0096:              Filename1 := 'test1.dat';
0097:           END;
0098:           IF x = 2 THEN BEGIN
0099:              n:=13;
0100:              s:=0.4;
0101:              h:=0.05;
0102:              Filename2 := 'C:\SW\carodata\SDH02_01.dat';
0103:              Filename1 := 'test2.dat';
0104:           END;
0105:           IF x = 3 THEN BEGIN
0106:              n:=14;
0107:              s:=1.0;
0108:              h:=0.05;
0109:              Filename2 := 'C: SW carodata SDH03_01.dat';
0110:              Filename1 := 'test3.dat';
0111:           END;
0112:           IF x = 4 THEN BEGIN
0113:              n:=15;
0114:              Filename2 := 'C: SW carodata SDH03_04.dat';
0115:              Filename1 := 'SDH3XH15.dat';
0116:           END;
0117:            IF x = 5 THEN BEGIN
0118:              n:=12;
0119:              Filename2 := 'C: SW carodata SDH15.dat';
0120:              Filename1 := 'SDH3XH15.dat';
0121:           END;
0122:           IF x = 6 THEN BEGIN
```

```
0123:              n:=13;
0124:              Filename2 := 'C:\SW\carodata\C0202_06.dat';
0125:              Filename1 := '2C25Nhi13.dat';
0126:           END;
0127:           IF x = 7 THEN BEGIN
0128:              n:=14;
0129:              Filename2 := 'C:\SW\carodata\C0202_07.dat';
0130:              Filename1 := '2C25Nhi14.dat';
0131:           END;
0132:           IF x = 8 THEN BEGIN
0133:              n:=15;
0134:              Filename2 := 'C:\SW\carodata\C0202_08.dat';
0135:              Filename1 := 'C25Nhi_15.dat';
0136:           END;
0137:           IF x = 9 THEN BEGIN
0138:           n:=12;
0139:              Filename2 := 'C:\SW\carodata\C0202_09.dat';
0140:              Filename1 := '2C25Nhi_12.dat';
0141:           END;
0142:           IF x =10 THEN BEGIN
0143:           n:=13;
0144:              Filename2 := 'C:\SW\carodata\C0202_10.dat';
0145:              Filename1 := '3C25Nhi_13.dat';
0146:           END;
0147:           IF x =11 THEN BEGIN
0148:           n:=14;
0149:              Filename2 := 'C:\SW\carodata\C0202_10.dat';
0150:              Filename1 := '3C25Nhi_14.dat';
0151:           END;
0152:           IF x =12 THEN BEGIN
0153:           n:=15;
0154:              Filename2 := 'C:\SW\carodata\C0202_10.dat';
0155:              Filename1 := '3C25Nhi_15.dat';
0156:           END;
0157:    IF DepFile THEN Assign(outfile,filename2);
0158:    IF MeanFile THEN Assign outfile,filename1);
0159:     Rewrite(outfile);
0160: END;
0161:                    {**********population**********}
0162:
0163:
0164: PROCEDURE InitFreq(VAR p0:AllelTable);
0165:     {This procedure assigns values to the initial allele-frequency p0
0166:      of allele a on every locus 1.}
0167:
0168: BEGIN
0169:    FOR l := 1 TO 16 DO
0170:       p0[l] := 0.1;
0171: {   For l := (n+1) to 16 do
0172:       p0[l] := 0.5;}
0173: END;     {of InitFreq}
0174:
0175:
0176: PROCEDURE InitPop (VAR Pop:population);
0177:     {This procedure initializes a population. It determines the genotype of
0178:      each individual. The genotype frequencies will be about Hardy-Weinberg
0179:      equilibrium.}
0180:
0181: BEGIN
0182:    FOR j := 1 TO popsize DO
0183:    WITH pop[j] DO
```

```
0184:    BEGIN
0185:       Geno[0] := 0;  Geno[1] := 0;
0186:       FOR l := 15 DOWNTO 0 DO
0187:       BEGIN
0188:         IF random < p0[16-l] THEN Geno[0] := Geno[0] + ( 1 SHL l);
0189:         IF random < p0[16-l] THEN Geno[1] := Geno[1] + ( 1 SHL l);
0190:       END;
0191:    END;
0192: END;           { of InitPop}
0193:
0194:
0195: PROCEDURE InitVMatrix;
0196:    {This procedure initializes a matrix [loci, genotype] of fitness-values.}
0197:
0198: BEGIN
0199: {   s := 0;
0200:    h := 0.01;}
0201:    FOR l := 1 TO 16 DO
0202:    BEGIN
0203:       Vmat[1,0] := 1;    VMat[1,1] := 1-h*s;    VMat[1,2] :=1-s;
0204:    END;
0205: {   For l := (n+1) to 16 do
0206:    begin
0207:       VMat[1,0]:=0.95;.      VMat[1,1] :=1;      VMat[1,2] :=0.95;
0208:    end;}
0209: END;     {of InitVMat}
0210:
0211:
0212:
0213: PROCEDURE ClearGenoFreq (VAR GenoFreq: FreqTable);
0214:     {Clears table of genotypes for next generation.}
0215:
0216: BEGIN
0217:    FOR l := 1 TO 16 DO
0218:       FOR k := 0 TO 2 DO
0219:          GenoFreq[l,k] := 0;
0220: END;       { of ClearGenoFreq}
0221:
0222:
0223:               {* * * * * * * * * * * * * * * * * * }
0224:
0225:
0226: PROCEDURE InitGeneration (VAR p0:AllelTable: VAR Pop:Population;
0227:                          VAR GenoFreq:FreqTable);
0228:     {This procedure initializes the first generation, calculates
0229:      its viability/fertility and writes the genotypes to a table.}
0230:
0231: VAR
0232:    Vint   :real;
0233:  { Fint   :real; }
0234:    lgeno  :ARRAY [1..16] OF byte;         {local genotype of individual}
0235:
0236: BEGIN
0237:    InitPop(Pop);
0238:    ClearGenoFreq(GenoFreq);
0239:
0240:    FOR j := 1 TO popsize DO               {calculate viability/fertility}
0241:    BEGIN                                  {of individual               }
0242:       Vint :=1;
0243:     { Fint :=1; }
0244:       FOR l:= 15 DOWNTO 0 DO
```

```
0245:       BEGIN
0246:         lgeno[16-l] := ((Pop[j].Geno[0] SHR l) AND 1) + ((Pop[j].Geno[1] SHR l) AND 1);
0247:         Vint := Vint * VMat[16-l,lgeno[16-l]];
0248:       { Fint := Fint * FMat[16-l,lgeno[16-l]];}
0249:         Inc (GenoFreq[16-l, lgeno[16-l]]);
0250:       END;
0251:        Pop[j].Viability := Vint; {Indi.Fertility :=Fint;}
0252:    END;
0253: END;       {of initGeneration}
0254:
0255:                 {* * * * * * * * * * * * * * * * * }
0256:
0257:
0258: FUNCTION CalcHetero(Genofreq: FreqTable) : real;
0259:    { This function calculates the mean frequency of heterozygotes
0260:      per locus in the present population. }
0261:
0262: VAR
0263:    Hint : integer;
0264:
0265: BEGIN
0266:    Hint := 0;
0267:    FOR l := 1 TO 16 DO
0268:       Hint := Hint + GenoFreq [l,1];
0269:    CalcHetero := Hint / (16*popsize);
0270: END;            { of CalcHetero}
0271:
0272:
0273: FUNCTION CalcPopViab(Pop : Population): real;
0274:    { This function calculates the mean viability of the population.}
0275:
0276: VAR
0277:    PopV : real;
0278:    j    : integer;
0279:
0280: BEGIN
0281:    PopV := 0;
0282:    FOR j := 1 TO popsize DO
0283:       PopV := PopV + Pop[j].Viability;
0284:    CalcPopViab := PopV/Popsize;
0285: END;     { of CalcPopViab}
0286:
0287:                       {* * * * * * *}
0288:
0289: PROCEDURE PopOOutput(Pop:Population; GenoFreq:FreqTable);
0290:      {Output to screen for generation 0.}
0291:
0292: BEGIN
0293:    Writeln('Generation ',gen);
0294:    PopViab[gen,sim] := CalcPopViab(Pop);
0295:    ActViab[gen,sim] := PopViab[gen,sim];
0296:    Hetero := CalcHetero(GenoFreq);
0297:    ActPopsize := popsize;
0298:    Writeln ('Population viability = ',PopViab[gen,sim]:5:3);
0299:    Writeln ('Frequency heterozygotes = ',Hetero:5:3);
0300:    writeln;
0301:    FOR l := 1 TO 16 DO                          {check fixation}
0302:    BEGIN
0303:       IF (GenoFreq[l,2] = popsize) THEN
0304:          Write('+  ') ELSE                        {a fixed : +}
0305:       IF (GenoFreq[l,0] = popsize)  THEN
```

```
0306:          Write('-  ') ELSE                          {A fixed : -}
0307:          write('0  ');
0308:     END;
0309:     Writeln;
0310: END;    {of PopOOutput}
0311:
0312:
0313: PROCEDURE PopOutput(Pop:Population; ActGenoFreq,GenoFreq:FreqTable);
0314:       {Output to screen generation 1 to popsize.}
0315:
0316: BEGIN
0317:    GOTOXY(1,2);
0318:    Writeln('Generation ',gen);
0319:    Writeln('ActPopsize = ',actpopsize,'    ');
0320:    PopViab[gen,sim] := CalcPopViab(Pop);
0321:    Hetero := CalcHetero(GenoFreq);
0322:    Writeln ('Zygote viability = ',ActViab[gen,sim]:5:3,'      ');
0323:    Writeln ('Population viability = ',PopViab[gen,sim]:5:3,'      ');
0324:    Writeln ('Frequency heterozygotes = ',Hetero:5:3,'      ');
0325:    FOR l := 1 TO 16 DO                          {check fixation}
0326:    BEGIN
0327:       IF (GenoFreq[l,2] = popsize) THEN
0328:          Write('+  ') ELSE                          {a fixed : +}
0329:       IF (GenoFreq[l,0] = popsize) THEN
0330:          Write('-  ') ELSE                          {A fixed : -}
0331:          Write('0  ');
0332:    END;
0333:    Writeln;
0334: END;    {of PopOutput}
0335:
0336:
0337:                 {************Zygote*************}
0338:
0339:
0340: PROCEDURE ChooseParent (Pop:population;VAR Parent:Individual);
0341:    {This procedure draws one random individual to serve as parent.}
0342:
0343: VAR
0344:    rand  : integer;
0345:
0346: BEGIN
0347:    rand := random (popsize) + 1;
0348:    Parent := Pop[rand];
0349: END;      {of ChooseParent}
0350:
0351:
0352: PROCEDURE ExtractHaplotype (Diplo: Diplotype; VAR Haplo: Haplotype);
0353:    { This procedure extracts a random haplotype (of sixteen loci)
0354:      from the genotype of one parent}
0355:
0356: VAR
0357:   rand: Word;
0358:
0359: BEGIN
0360:    rand:= random(65535);
0361:    Haplo:= (rand AND Diplo[0]) OR ((65535-rand) AND Diplo[1]);
0362: END; {of ExtractHaplotype}
0363:
0364:
0365:
0366: PROCEDURE Mutation(VAR Haplo:Haplotype);
```

```
0367:    { This procedure allows for mutation from A to a with
0368:      U (= u*16) = 2.0E-4.}
0369:
0370: VAR
0371:    Mask   :word;
0372:    rand   :double;
0373:    loc    :word;
0374:
0375: BEGIN
0376:    rand := random;
0377:    IF rand < 2E-4 THEN                    {mutation ?}
0378:    BEGIN
0379:       loc := random(16)  ;                {where ?}
0380:       IF NOT(odd(Haplo SHR loc)) THEN     {Allele is A ?}
0381:       BEGIN
0382:          Mask := 1 SHL loc;
0383:          Haplo := Haplo OR Mask;     {0 becomes 1, i.e. A becomes a}
0384:       END;
0385:    END;
0386: END;   {of Mutation}
0387:
0388:                  {* * * * * * * * * * * * * * * * * *}
0389:
0390: PROCEDURE ConstructZygote(VAR Indi:Individual; ParentPop:Population);
0391:    {This procedure constructs the genotype of a new individual (zygote).}
0392:
0393: BEGIN
0394:    ChooseParent(ParentPop,Parent);
0395:    ExtractHaplotype(Parent.Geno,Haplo);
0396:    Mutation(Haplo);
0397:    Indi.Geno[0] := Haplo;
0398:    ChooseParent(ParentPop,Parent);
0399:    ExtractHaplotype(Parent.Geno,Haplo);
0400:    Mutation(Haplo);
0401:    Indi.Geno[1] := Haplo;
0402: END;   {of ConstructZygote}
0403:
0404:
0405:
0406: PROCEDURE Evaluate ( VAR Indi:Individual; VAR survive : boolean);
0407:    {This procedure calculates viability/fertility of an individual
0408:     and tests its survival. Then its genotype is written to the
0409:     genotype-table. It also keeps record of those that don't survive.}
0410:
0411: VAR
0412:    Vint   :real;
0413:  { Fint   :real; }
0414:    lgeno  :ARRAY [1..16] OF byte;         {local genotype of individual}
0415:
0416: BEGIN
0417:    Vint :=1;
0418:  { Fint :=1; }
0419:    FOR l:= 15 DOWNTO 0 DO
0420:    BEGIN
0421:      lgeno[16-l] :=  ((Indi.Geno[0] SHR l) AND 1) +((Indi.Geno[1] SHR l) AND 1);
0422:         Vint := Vint * VMat[16-l,lgeno[16-l]];
0423:       { Fint := Fint * FMat[16-l,lgeno[16-l]];}
0424:    Inc(ActGenoFreq[16-l, lgeno[16-l]]);
0425:    END;
0426:    Indi.Viability := Vint; {Indi.Fertility :=Fint;}
0427:    Inc(actPopsize);
```

```
0428:    IF random < Vint THEN survive:=True                {survives?, random is}
0429:                    ELSE survive:=False;               {bad luck number}
0430:    IF survive THEN
0431:    BEGIN
0432:       FOR l:=15 DOWNTO 0 DO  Inc (GenoFreq[16-l, lgeno[16-l]]);
0433:    END;
0434: END;    {of Evaluate}
0435:
0436:
0437:
0438: PROCEDURE PFreq(GenoFreq: FreqTable; VAR Ppd,Ph: TotArray);
0439:   {This procedure calculates the mean frequency of "a" on the dominant loci
0440:    and on the overdominant loci.}
0441:
0442: VAR
0443:    p    :ARRAY[1..16] OF single;
0444:    pint :single;
0445:
0446: BEGIN
0447:    FOR l := 1 TO 16 DO
0448:       p[l] := (GenoFreq[l,2] + 0.5*GenoFreq[l,1])/popsize;
0449:    pint := 0;
0450:    FOR l := 1 TO n DO
0451:       pint := pint + p[l];
0452:    Ppd[gen,sim] := pint/n;
0453:    pint := 0;
0454:    FOR l := (n+1) TO 16 DO
0455:       pint := pint + p[l];
0456:    Ph[gen,sim] := pint/(16-n);
0457: END;
0458:
0459:                    (* * * * * * * * * * * * * * * *)
0460:
0461: PROCEDURE NextGeneration (VAR GenoFreq:FreqTable; VAR Pop2: Population);
0462:      {This procedure constructs the individuals of the next generation,
0463:       evaluates them and replaces them if they do not survive. They are
0464:       saved as Pop2.}
0465:
0466: VAR
0467:   j       : integer;
0468:
0469: BEGIN
0470:    ClearGenoFreq(GenoFreq);
0471:    ClearGenoFreq(ActGenoFreq);
0472:    ActViab[gen,sim] :=0;
0473:    ActPopsize :=0;
0474:    FOR j := 1 TO popsize DO
0475:    BEGIN
0476:       survive := False;
0477:       REPEAT
0478:          ConstructZygote (Pop2[j],Pop);
0479:          Evaluate (Pop2[j],Survive);
0480:          ActViab[gen,sim] := ActViab[gen,sim] + Pop2[j].viability;
0481:       UNTIL survive;
0482:    END;
0483:    ActViab[gen,sim] := ActViab[gen,sim]/ActPopsize;
0484:    END;          {of NextGeneration}
0485:
0486:
0487:
0488:              {*****************Generation*****************}
```

```
0489:
0490:
0491: PROCEDURE OneGeneration(VAR Pop,Pop2: Population; VAR GenoFreq: FreqTable);
0492:     {This procedure calculates the data of the present generation,
0493:      creates the next generation and replaces the present population
0494:      by the next.}
0495:
0496: BEGIN
0497:     NextGeneration (GenoFreq,Pop2);
0498:     PFreq(GenoFreq,Ppd,Ph);
0499:     Pop:= Pop2;                              {replace old by new generation}
0500:     PopViab[gen,sim] := CalcPopViab(Pop);
0501:     Hetero := CalcHetero(GenoFreq);
0502:     PopOutPut(Pop,ActGenoFreq,GenoFreq);
0503: END;           {of OneGeneration}
0504:
0505:
0506:               {*****************File******************}
0507:
0508:
0509: PROCEDURE InputToFile(MeanFile:boolean);
0510:
0511: VAR
0512:    l : integer;
0513:
0514: BEGIN
0515:   IF MeanFile THEN
0516:   BEGIN
0517:      Writeln(outfile,Filename1);
0518:      Writeln(outfile);
0519:      Writeln(outfile, 'popsize = ',popsize:4);
0520:      Writeln(outfile);
0521:      Writeln(outfile,'p0    , fitness matrix');
0522:      FOR l := 1 TO 16 DO
0523:         Writeln(outfile,p0[l]:4:3,' , ',VMat[l,0]:4:3,'   ',VMat[l,1]:4:3,'   ',Vmat[l,2]:4:3);
0524:      Writeln(outfile);
0525:   END;
0526: END;   {of InputToFile}
0527:
0528:
0529:
0530: PROCEDURE MeanToStr (VAR MeanStr:str70);
0531:    {This procedure writes the viability-means to a string
0532:     for storing in file.}
0533:
0534: VAR
0535:    st0 ,StGen                        :STRING[3];
0536:    stZMean,stPMean,stInf,stD,stDS    :STRING[6];
0537:    stZS,stPS,stPpd, stPh             :STRING[6];
0538:
0539: BEGIN
0540:    st0:='   ';
0541:    Str(Gen:3,StGen);
0542:    Str(ActMean[gen]:6:3,StZMean);       Str(ActS[gen]:6:3,stZS);
0543: (*   Str(PopMean[gen]:6:3,stPMean);       Str(PopS[gen]:6:3,stPS);*)
0544:    Str(Wtot[gen]:6:3,stInf);
0545:    Str(D[gen]:6:3,stD);                 Str(DS[gen]:6:3,stDS);
0546:    Str(PpdMean[gen]:6:3,stPpd);
0547:    Str(PhMean[gen]:6:3,stPh);
0548:    MeanStr := Concat (StGen,st0,stZMean,st0,stZS,st0,stInf,st0,stD,
0549:                       st0,StDS,st0,stPpd,st0,stPh);
```

```
0550: END;   {of MeanToStr}
0551:
0552:
0553: PROCEDURE  DepToStr (VAR Depstr:str70);
0554:
0555: VAR
0556:   st0 ,stGen          :STRING[3];
0557:   stDep,stDS          :STRING[6];
0558:
0559: BEGIN
0560:    st0 :='   ';
0561:    Str(Gen:3,stGen);
0562:    Str(D[gen]:6:3,stDep);       {Dw lopend gem.
0563:    Str(DS[gen]:6:3,stDS);
0564:    DepStr:= concat(stGen,st0,stDep,st0,stDS);
0565: END;
0566:
0567:
0568:                {*************Infinite*************}
0569:
0570:
0571: PROCEDURE Infinite(VMat:Matrix; VAR Wtot:MeanArray);
0572:    {This procedure calculates the viability of an infinite population
0573:     for the same selection and initial frequencies as Pop.}
0574:
0575: VAR
0576:    Wint    :real;
0577:    W       :ARRAY [1..16] OF single;
0578:
0579: BEGIN
0580:    InitFreq(pa2);
0581:    FOR gen := 0 TO generation DO
0582:    BEGIN
0583:       Wint := 1;
0584:       FOR i := 1 TO 16 DO
0585:       BEGIN
0586:          pa[i] := pa2[i];
0587:          W[i] :=  SQR(1-pa[i]) * VMat[1,0] + 2 * pa[i] * (1-pa[i]) * VMat[1,1]
0588:                + (SQR(pa[i]) * VMat[1,2]);
0589:          Wint := Wint * W[i];
0590:          pa[i] := pa[i] * (pa[i] * VMat[1,0] + (1-pa[i]) * VMat[1,1]) / W[i];
0591:       END;
0592:       Wtot[gen] := Wint;
0593:    END;
0594: END;     {of Infinite
0595:
0596:                {*******************Over All*******************}
0597:
0598:
0599: PROCEDURE CalcMean (Tab:TotArray; VAR Mean,S:MeanArray);
0600:    {This procedure calculates the aritmic mean and standard deviation
0601:     of a given parameter over the simulations.
0602:
0603: VAR
0604:    Int,Varint  : real;
0605:    sim         : integer;
0606:
0607: BEGIN
0608:    FOR gen := 0 TO generation DO
0609:    BEGIN
0610:       Int := 0;
```

```
0611:       Varint := 0;
0612:       FOR simu := 1 TO simulation DO                          {mean}
0613:          Int := Int + Viab[gen,simu];
0614:       Mean[gen] := Int / simulation;
0615:       FOR simu := 1 TO simulation DO                          {st.deviation}
0616:          Varint := Varint + SQR(Viab[gen,simu] - Mean[gen]);
0617:       S[gen] := SQRT(Varint/simulation);
0618:    END;
0619: END;    {of CalcMean}
0620:
0621:
0622: PROCEDURE MeanSmall(Viab:TotArray; VAR Mean:MeanArray);
0623:   {This procedure calculates mean without standard deviation.}
0624:
0625: VAR
0626:    Int,Varint  : real;
0627:    simu        : integer;
0628:
0629: BEGIN
0630:    FOR gen := 0 TO generation DO
0631:    BEGIN
0632:       Int := 0;
0633:       Varint := 0;
0634:       FOR simu := 1 TO simulation DO                          {mean}
0635:          Int := Int + Viab[gen,simu];
0636:       Mean[gen] := Int / simulation;
0637:    END;
0638: END;
0639:
0640:
0641: PROCEDURE  MeanAll;
0642:    {This procedure calculates the mean and standard deviation of
0643:     two viability measures and the viability of an infinite population
0644:     and sends them to file.}
0645:
0646: BEGIN
0647:    CalcMean(PopViab,PopMean,PopS);
0648:    CalcMean(ActViab,ActMean,ActS);
0649:    MeanSmall(Ppd,PpdMean);
0650:    MeanSmall(Ph,PhMean);
0651:    Infinite(VMat,Wtot);
0652: END;     {of MeanAll}
0653:
0654:
0655: PROCEDURE CalcDepression(VAR D,Dw:MeanArray);
0656:     {This procedure calculates the viability of the finite population
0657:      (Pop) relative to the infinite population.}
0658:
0659: VAR
0660:    a,b,c   :integer;
0661:    Dint    :real;
0662:    Dt      :ARRAY [9..generation] OF real;
0663:
0664: BEGIN
0665:    MeanAll;
0666:    FOR gen := 0 TO generation DO
0667:    BEGIN
0668:        D[gen]  := 1 - ActMean[gen]/Wtot[gen];
0669:        DS[gen] := ActS[gen]/Wtot[gen];
0670:    END;
0671:
```

```
0672: (*   If DepFile then
0673:    begin
0674:      Dint := 0;                              {walking mean over 10 generations}
0675:      FOR a := 5 TO 14 DO                      {??????????????????????????????????}
0676:      Dint := Dint + D[a];
0677:      Dt[14] := Dint;
0678:      FOR b := 14 TO (generation-1) DO
0679:      Dt[b+1] := Dt[b] + D[b+1] - D[b-9];
0680:      FOR c := 0 TO 14 DO
0681:      Dw[c] := Dt[14]/10;
0682:      FOR c := 15 TO generation DO
0683:      Dw[c] := Dt[c]/10;
0684:    END;  *)
0685:
0686:    FOR gen := 0 TO generation DO           {write to file}
0687:    BEGIN
0688:      IF DepFile THEN
0689:      BEGIN
0690:        DepToStr(DepStr);
0691:        Writeln(outfile, Depstr);
0692:      END;
0693:   END;
0694: END;       {of CalcDepression}
0695:
0696:
0697: PROCEDURE EndFix (GenFreq:FreqTable);
0698:      {This procedure writes fixations in last generation to file.}
0699:
0700: BEGIN
0701:    IF MeanFile THEN
0702:    BEGIN
0703:       Write(outfile,'simulation ',sim,'   ');
0704:       FOR l := 1 TO 16 DO                                  {check fixation}
0705:       BEGIN
0706:          IF  GenoFreq[l,2] = popsize) THEN
0707:             Write(outfile,'a  ') ELSE                          {a fixed : a}
0708:          IF (GenoFreq[l,3] = popsize) THEN
0709:             Write(outfile,'A  ') ELSE                          {A fixed : A}
0710:             Write outfile,'?  ');
0711:       END;
0712:       Writeln outfile;
0713:    END;
0714: END;    {of EndFix}
0715:
0716:
0717:
0718:                 * * * * * * * * *Compare* * * * * * * * *
0719:
0720: PROCEDURE EndFreq GenoFreq:FreqTable; VAR Pend:PTable ;
0721:       {This procedure writes pA in the last generation to an array.
0722:
0723: BEGIN
0724:    FOR l := 1 TO 16 DO
0725:    Pend[sim,l] := (GenoFreq[l,2] + 0.5*GenoFreq[l,1]) popsize;
0726: END;    {of EndFreq
0727:
0728:
0729: PROCEDURE GeneDiffer Pend: PTable; VAR Gst:single ;
0731:       This procedure computes Gst, the coefficient of gene-differentiation,
0731:       based on the real frequency of A in the last generation of each simulation.
0732:
```

```
0733: VAR
0734:    Pint, Hsint, Hs, Ht, Fst  : ARRAY[1..16] OF real;
0735:    Gstint                    : real;
0736:
0737: BEGIN
0738:    FOR l := 1 TO 16 DO
0739:    BEGIN
0740:       Pint[l] := 0;     Hsint[l] := 0;      Gstint := 0;
0741:       FOR sim := 1 TO simulation DO
0742:       BEGIN
0743:          Pint[l]:= Pint[l] + Pend[sim,l];           {mean pA over simulations}
0744:          Hsint[l] := Hsint[l] + (SQR(Pend[sim,l]) + SQR(1- Pend[sim,l]));
0745:                                                     {exp. homo. per simulation}
0746:       END;
0747:       Pint[l] := Pint[l]/simulation;                {mean pA}
0748:       Hs[l] := 1 - Hsint[l]/simulation;             {mean exp. heterozygosity}
0749:       Ht[l] := 1 - (SQR(Pint[l]) + SQR(1-Pint[l])); {expected heterozygosity all}
0750:       IF Ht[l] = 0 THEN Fst[l]:= 0 ELSE
0751:       BEGIN
0752:          Fst[l] := 1 - (Hs[l]/Ht[l]);               {for one locus}
0753:          Gstint := Gstint + Fst[l];
0754:       END;
0755:    END;
0756:       Gst := Gstint/16;                             {for all 16 loci}
0757:    Writeln('Gst=',Gst:5:4);      {test}
0758: END;       {of GeneDiffer}
0759:
0760:
0761: PROCEDURE Distance(Pend:PTable);
0762:     {This procedure calculates genetic identity between the first 3
0763:      simulations according to M. Nei.}
0764:
0765: VAR
0766:    XX, XY                    :ARRAY[1..3, 1..16] OF real;
0767:    XXint, XYint, Jxx, Jxy    :ARRAY[1..3] OF real;
0768:
0769: BEGIN
0770:    FOR sim := 1 TO 3 DO                {calculate Jxx for sim 1,2,3}
0771:    BEGIN
0772:       XXint[sim] :=0;
0773:       FOR l := 1 TO 16 DO
0774:       BEGIN
0775:          XX[sim,l] := SQR(Pend[sim,l]) + SQR(1-Pend[sim,l]);
0776:          XXint[sim] := XXint[sim] + XX[sim,l];
0777:       END;
0778:       Jxx [sim] := XXint[sim]/16;                    {for all loci}
0779:    END;
0780:    FOR l := 1 TO 16 DO              {calculate Jxy for 1-2, 1-3, 2-3}
0781:    BEGIN
0782:       XY[1,l] := Pend[1,l] * Pend[2,l]  + (1-Pend[1,l]) * (1-Pend[2,l]);
0783:       XY[2,l] := Pend[1,l] * Pend[3,l]  + (1-Pend[1,l]) * (1-Pend[3,l]);
0784:       XY[3,l] := Pend[2,l] * Pend[3,l]  + (1-Pend[2,l]) * (1-Pend[3,l]);
0785:    END;
0786:    FOR sim := 1 TO 3 DO
0787:    BEGIN
0788:       XYint[sim] := 0;
0789:       FOR l := 1 TO 16 DO
0790:          XYint[sim] := XYint[sim] + XY[sim,l];
0791:       Jxy[sim] := XYint[sim]/16;                     {for all loci}
0792:    END;
0793:    I[1] :=  Jxy[1]/SQRT(Jxx[1]*Jxx[2]);
```

```
0794:    Write('I(1-2) =',I[1]:5:3,'  ');                              {test}
0795:    I[2] :=  Cxy[2]/SQRT(Cxx[1]*Cxx[3]);
0796:    Write('I(1-3) =',I[2]:5:3,'  ');                              {test}
0797:    I[3] :=  Cxy[3]/SQRT(Cxx[2]*Cxx[3]);
0798:    Writeln('I(2-3) =',I[3]:5:3);                                 {test}
0799: END;     {of Distance}
0800:
0801:
0802: PROCEDURE AllOutput(MeanFile:boolean);
0803:     {This procedure sends Gst and genetic identity to file,
0804:      as well as the viability means and st.deviations.}
0805:
0806: BEGIN
0807:    IF MeanFile THEN
0808:    BEGIN
0809:       Writeln(outfile);
0810:       Writeln(outfile,' gen.  ZygoteV    s      infV     Depinf    s     pPD     pH');
0811:       FOR gen := 0 TO 9 DO
0812:       BEGIN
0813:          MeanToStr(MeanStr);
0814:          Writeln(outfile,MeanStr);
0815:       END;
0816:       gen := 10;
0817:       REPEAT
0818:          MeanToStr(MeanStr);
0819:          Writeln(outfile,MeanStr);
0820:          gen:= gen + 5;
0821:       UNTIL gen > generation;
0822:       Writeln(outfile);
0823:       Writeln(outfile,'Gst = ',Gst:6:4);  {write to file}
0824:       Write(outfile,'I(1-2) = ',I[1]:5:3,'   ');
0825:       Write outfile,'I(1-3) = ',I[2]:5:3,'   ');
0826:       Writeln(outfile,'I(2-3) = ',I[3]:5:3);
0827:    END;
0828: END;
0829:
0830: {****************************** Main Program ******************************}
0831:
0832: BEGIN
0833:    FOR x := 1 TO 3 DO
0834:    BEGIN
0835:       OpenFile;
0836:       Randomize;
0837:       InitFreq(p0);
0838:       InitVMatrix;
0839:       InputToFile(MeanFile);
0840:       sim := 1;
0841:       WHILE sim <= simulation DO
0842:       BEGIN
0843:          ClrScr;
0844:          Writeln 'X = ',x,'  SIMULATION ',sim;  Writeln;
0845:          InitGeneration(p0,Pop,GenoFreq);          {Create generation 0}
0846:          gen := 0;
0847:          Pop0CntPut Pop,GenoFreq);
0848:          WHILE gen < generation DO
0849:          BEGIN
0850:            gen := gen + 1;                    {replace population}
0851:            NxtGeneration Pop,Pop,GenoFreq);   {and evaluate}
0852:          END;
0853:          EndFix GenoFreq );
0854:          EndFreq GenoFreq,Pen);
```

```
0855:          { Write(#7); Delay(4000);}                    {Beep, wait 4 sec}
0856:           sim:= sim+1;
0857:        END;
0858:        CalcDepression(D,Dw);
0859:        GeneDiffer(Pend,Gst);
0860:        Distance(Pend);
0861:        AllOutput(MeanFile);
0862:        Close(outfile);
0863:    END;
0864:    REPEAT UNTIL KeyPressed;
0865: END.
0866:
0867: (************************************END**********************************)
0868:
0869:
0870:
```

Appendix 2: Source code of computer programme Inbreed3.pas (model 3 + 1).

```
0001:
0002: PROGRAM Inbreed3;
0003:
0004:     ***************************************************************************
0005:     * This program simulates the effects of inbreeding and selection on     *
0006:     * population-fitness, genotype-frequencies and populationsize           *
0007:     * in small, non-selfing populations.                                    *
0008:     * There are 16 uncoupled loci. Each locus has two possible alleles :    *
0009:     * a and A. The loci gouvern viability.                                  *
0010:     * The fitness of an individual is the product of its fitness at         *
0011:     * every locus. Mutations are allowed for. The generations are           *
0012:     * seperated.                                                            *
0013:     * Allele-frequencies and selection may be modified in the program,      *
0014:     * as well as the mean number of seeds per parent.                       *
0015:     * Population size varies over the generations depending on the          *
0016:     * viability of the zygotes.                                             *
0017:     * -Carolien de Kovel-                                                   *
0018:     ***************************************************************************|
0019:
0020: uses Crt;
0021:
0022: CONST
0023:    seed       = 1.5;                    {mean number of seeds per plant}
0024:    capacity   = 50;                     {carrying capacity of habitat}
0025:    generation = 50;
0026:    simulation = 5;
0027:    DataFile   = True;                        {data 1 simulation to file}
0028:    MeanFile   = True;                           {write means to file ?}
0029:    DepFile    = False;                      {write depression to slide ?}
0030:    Filename1:STRING = 'file1';
0031:    Filename2:STRING = 'file2';
0032:
0033: TYPE
0034:     Haplotype  = word;
0035:     Diplotype  = ARRAY [0..1] OF Haplotype;
0036:     FreqTable  = ARRAY [1..16, 0..2] OF integer;
0037:     Individual = RECORD
0038:                    Geno     : Diplotype;
0039:                    Viability: real;
0040:                    Fertility: real;
0041:                    END;
0042:     Population = ARRAY [1..capacity] OF Individual;
0043:     AllelTable = ARRAY [1..16] OF single;
0044:     QArray     = ARRAY [0..10] OF single;
0045:     Matrix     = ARRAY [1..16, 0..2] OF single;
0046:     str70      = STRING[70];
0047:     PTable     = ARRAY [1..simulation, 1..16] OF single;
0048:     TotArray   = ARRAY [0..generation, 1..simulation] OF  single;
0049:     MeanArray  = ARRAY [0..generation] OF single;
0050:
0051: VAR
0052:     {Act... is popsize, viability etc. before selection}
0053:     j,k,gen,sim,z              :integer;
0054:     popsize                    :integer;
0055:     l                          :byte;
0056:     Hetero, Gst                :real;
0057:     Viability, Fertility       :single;
0058:     s,h,u                      :single;
0059:     Survive                    :boolean;
0060:     Haplo                      :Haplotype;
0061:     Diplo                      :Diplotype;
```

```
0062:     VMat                    :Matrix ;
0063:     GenoFreq,ActGenoFreq    :FreqTable;
0064:     Indi, parent            :Individual;
0065:     Pop, Pop2               :Population;
0066:     p0, Fst, pa, pa2        :AllelTable;
0067:     Pend                    :PTable;     {mean pA in last generation}
0068:     Datastr,Meanstr,DepStr  :str70;
0069:     PopStr                  :str70;
0070:     PopViab,ActViab         :TotArray;
0071:     PopMean,ActMean         :MeanArray;
0072:     PopS,ActS,DS            :MeanArray;
0073:     Wtot,D,Dw               :MeanArray;
0074:     Q                       :QArray;
0075:     outfile                 :text;
0076:     ActPopsize              :integer;
0077:     I                       :ARRAY[1..3] OF single;
0078:
0079:                   {*****************************}
0080:
0081: PROCEDURE WriteWord (w : Word; wstr : STRING);
0082:     {Write binary code of w}
0083:
0084: BEGIN
0085:    Write(wstr,' =   ');
0086:    FOR j := 15 DOWNTO 0 DO
0087:     IF Odd(w SHR j) THEN Write ('1 ')
0088:                     ELSE write('0 ');
0089: END;     {of WriteWord}
0090:
0091:
0092: PROCEDURE OpenFile;
0093:     {This procedure opens files for various simulations in Slide
0094:      or TP, not both!}
0095:
0096: BEGIN
0097:    IF DepFile AND MeanFile THEN Halt;
0098:          IF x = 1 THEN BEGIN
0099:             Filename2 := 'C:\slide\carodata\vartest.dat';
0100:             Filename1 := 'vartest1.dat';
0101:          END;
0102:          IF x = 2 THEN BEGIN
0103:             Filename2 := 'C:\slide\carodata\vartest2.dat';
0104:             Filename1 := 'vartest2.dat';
0105:          END;
0106:          IF x = 3 THEN BEGIN
0107:             Filename2 := 'C:\slide\carodata\vartest3.dat';
0108:             Filename1 := 'vartest3.car';
0109:          END;
0110:          IF x = 4 THEN BEGIN
0111:             Filename2 := 'C:\slide\carodata\vartest4.dat';
0112:             Filename1 := 'vartest4.car';
0113:          END;
0114:           IF x = 5 THEN BEGIN
0115:             Filename2 := 'C:\Slide\carodata\vartest5.dat';
0116:             Filename1 := 'vartest5.car';
0117:          END;
0118:          IF x = 6 THEN BEGIN
0119:             Filename2 := 'C:\slide\carodata\vartest6.dat':
0120:             Filename1 := 'vartest6.car';
0121:          END;
0122:          IF x = 7 THEN BEGIN
```

```
0123:             Filename2 := 'C:\slide\carodata\vartest7.dat';
0124:             Filename1 := 'vartest7.car';
0125:           END;
0126:           IF x = 8 THEN BEGIN
0127:             Filename2 := 'C:\slide\carodata\vartest8.dat';
0128:             Filename1 := 'vartest8.car';
0129:           END;
0130:           IF x = 9 THEN BEGIN
0131:             Filename2 := 'C:\slide\carodata\vartest9.dat';
0132:             Filename1 := 'vartest9.car';
0133:           END;
0134:           IF x = 10 THEN BEGIN
0135:             Filename2 := 'C:\slide\carodata\vartest0.dat';
0136:             Filename1 := 'vartest0.car';
0137:           END;
0138:           IF x = 11 THEN BEGIN
0139:             Filename2 := 'C:\slide\carodata\test9.dat';
0140:             Filename1 := 'test9.car';
0141:           END;
0142:           IF x = 12 THEN BEGIN
0143:             Filename2 := 'C:\slide\carodata\Hp25_10.dat';
0144:             Filename1 := 'Hp25_10.car';
0145:           END;
0146:     IF DepFile (or Datafile) THEN Assign(outfile,filename2);
0147:     IF MeanFile OR Datafile THEN Assign(outfile,filename1);
0148:     Rewrite(outfile);
0149: END;
0150:                       {**********population***********}
0151:
0152:
0153: PROCEDURE InitFreq(VAR p0:AlleleTable);
0154:     {This procedure assigns values to the initial allele-frequency p0
0155:      of allele a on every locus l.}
0156:
0157: BEGIN
0158:     FOR l := 1 TO 16 DO
0159:       p0[l] := 0.1;
0160: {    For l := 4 to 16 do
0161:       p0[l] := 0.5;}
0162: END;      {of InitFreq}
0163:
0164:
0166: PROCEDURE InitPop  VAR Pop:population);
0166:     {This procedure initializes a population. It determines the genotype of
0167:      each individual. The genotype frequencies will be about Hardy-Weinberg
0168:      equilibrium.
0169:
0170: BEGIN
0171:     popsize := 10;
0172:     FOR j := 1 TO popsize DO
0173:     WITH pop[j] DO
0174:     BEGIN
0175:       Geno[0] := 0;  Geno[1] := 0;
0176:       FOR l := 16 DOWNTO 0 DO
0177:       BEGIN
0178:         IF random < p0[16-l] THEN Geno[0] := Geno[0] +   1 SHL l;
0179:         IF random < p0[16-l] THEN Geno[1] := Geno[1] +   1 SHL l;
0180:       END;
0181:     END;
0182: END;             {of InitPop
0183:
```

```
0184:
0185: PROCEDURE InitVMatrix;
0186:    {This procedure initializes a matrix [loci, genotype] of fitness-values.}
0187:
0188: BEGIN
0189:    s := 0.4;
0190:    h := 0.05;
0191:    FOR l := 1 TO 16 DO
0192:    BEGIN
0193:       Vmat[l,0] := 0.95;    VMat[l,1] := 1;    VMat[l,2] :=0.95;
0194:    END;
0195: {   For l := 4 to 16 do
0196:    begin
0197:       Vmat[l,0] := 1;    VMat[l,1] := 1-h*s;    VMat[l,2] :=1-s;
0198:    end;}
0199: END;     {of InitVMat}
0200:
0201:
0202:
0203: PROCEDURE ClearGenoFreq (VAR GenoFreq: FreqTable);
0204:     {Clears table of genotypes for next generation.}
0205:
0206: BEGIN
0207:    FOR l := 1 TO 16 DO
0208:       FOR k := 0 TO 2 DO
0209:          GenoFreq[l,k] := 0;
0210: END;       { of ClearGenoFreq}
0211:
0212:
0213:                {* * * * * * * * * * * * * * * * * * }
0214:
0215:
0216: PROCEDURE InitGeneration (VAR p0:AllelTable; VAR Pop:Population;
0217:                           VAR GenoFreq:FreqTable);
0218:     {This procedure initializes the first generation, calculates
0219:      its viability/fertility and writes the genotypes to a table.}
0220:
0221: VAR
0222:    Vint   :real;
0223:  { Fint   :real; }
0224:    lgeno  :ARRAY [1..16] OF byte;         {local genotype of individual}
0225:
0226: BEGIN
0227:    InitPop(Pop);
0228:    ClearGenoFreq(GenoFreq);
0229:    FOR j := 1 TO popsize DO               {calculate viability/fertility}
0230:    BEGIN                                  {of individual               }
0231:       Vint :=1;
0232:     { Fint :=1; }
0233:       FOR l:= 15 DOWNTO 0 DO
0234:       BEGIN
0235:         lgeno[16-l] := ((Pop[j].Geno[0] SHR l) AND 1) + ((Pop[j].Geno[1] SHR l) AND 1);
0236:         Vint := Vint * VMat[16-l,lgeno[16-l]];
0237:       { Fint := Fint * FMat[16-l,lgeno[16-l]];}
0238:         Inc (GenoFreq[16-l, lgeno[16-l]]);
0239:       END;
0240:       Pop[j].Viability := Vint; {Indi.Fertility :=Fint;}
0241:    END;
0242: END;       {of initGeneration}
0243:
0244:                {* * * * * * * * * * * * * * * * * * }
```

```
0245:
0246:
0247: FUNCTION CalcHetero(GenoFreq: FreqTable  : real:
0248:    { This function calculates the mean frequency of heterozygotes
0249:      per locus in the present population. }
0250:
0251: VAR
0252:    Hint : integer:
0253:
0254: BEGIN
0255:    Hint := 0:
0256:    FOR i := 1 TO 16 DO
0257:       Hint := Hint + GenoFreq [i,1]:
0258:    CalcHetero := Hint / (16*popsize):
0259: END:            { of CalcHetero:
0260:
0261:
0262: FUNCTION CalcPopViab(Pop : Population): real;
0263:    { This function calculates the mean viability of the population.}
0264:
0265: VAR
0266:    PopV : real:
0267:    j    : integer:
0268:
0269: BEGIN
0270:    PopV := 0:
0271:    FOR j := 1 TO popsize DO
0272:       PopV := PopV + Pop[j].Viability:
0273:    CalcPopViab := PopV/Popsize:
0274: END:     { of CalcPopViab:
0275:
0276:                          * * * * * * *}
0277:
0278: PROCEDURE PopOutput(Pop:Population: GenoFreq:FreqTable):
0279:       Output to screen for generation 0..
0280:
0281: BEGIN
0282:    Writeln('Generation ',gen):
0283:    Writeln:
0284:    PopViab[gen,sim] := CalcPopViab(Pop):
0285:    ActViab[gen,sim] := PopViab[gen,sim]:
0286:    Hetero := CalcHetero GenoFreq :
0287:    ActPopsize := popsize:
0288:    Writeln 'Popsize = ',popsize:
0289:    writeln:
0290:    Writeln  'Population viability = ' PopViab[gen sim]:3:3:
0291:    Writeln  'Frequency heterozygotes = ' Hetero:3:3:
0292:    FOR i := 1 TO 16 DO                              {check fixation:
0293:    BEGIN
0294:       IF  GenoFreq[i,2] = popsize  THEN
0295:          Write '- ' ELSE                             {A fixed : -
0296:       IF  GenoFreq[i,2] = popsize  THEN
0297:          Write '- ' ELSE                             {a fixed : -
0298:          write'i ' :
0299:    END:
0300:    Writeln:
0301: END:     {of PopOutput
0302:
0303:
0304: PROCEDURE PopOutput Pop:Population: ActGenoFreq,GenoFreq:FreqTable :
0305:      {Output to screen generation 1 to popsize.
```

```
0306:
0307: BEGIN
0308:    GOTOXY(1,2);
0309:    Writeln('Generation ',gen);
0310:    Writeln('ActPopsize = ',actpopsize,'    ');
0311:    Writeln ('Popsize = ', popsize,' ');
0312:    PopViab[gen,sim] := CalcPopViab(Pop);
0313:    Hetero := CalcHetero(GenoFreq);
0314:    Writeln ('Zygote viability = ',ActViab[gen,sim]:5:3);
0315:    Writeln ('Population viability = ',PopViab[gen,sim]:5:3);
0316:    Writeln ('Frequency heterozygotes = ',Hetero:5:3);
0317:    FOR l := 1 TO 16 DO                          {check fixation}
0318:    BEGIN
0319:       IF (GenoFreq[l,0] = popsize) THEN
0320:          Write('-  ') ELSE                        {A fixed : -}
0321:       IF (GenoFreq[l,2] = popsize) THEN
0322:          Write('+  ') ELSE                        {a fixed : +}
0323:          Write('0  ');
0324:    END;
0325:    Writeln;
0326: END;    {of PopOutput}
0327:
0328:
0329:                {************Zygote*************}
0330:
0331:
0332: PROCEDURE ChooseParent (Pop:population;VAR Parent:Individual);
0333:    {This procedure draws one random individual to serve as parent.}
0334:
0335: VAR
0336:    rand  : integer;
0337:
0338: BEGIN
0339:    rand := random (popsize) + 1;
0340:    Parent := Pop[rand];
0341: END;      {of ChooseParent}
0342:
0343:
0344: PROCEDURE ExtractHaplotype (Diplo: Diplotype; VAR Haplo: Haplotype);
0345:    { This procedure extracts a random haplotype (of sixteen loci)
0346:      from the genotype of one parent}
0347:
0348: VAR
0349:   rand: Word;
0350:
0351: BEGIN
0352:    rand:= random(65535);
0353:    Haplo:= (rand AND Diplo[0]) OR ((65535-rand) AND Diplo[1]);
0354: END; {of ExtractHaplotype}
0355:
0356:
0357:
0358: PROCEDURE Mutation(VAR Haplo:Haplotype);
0359:    { This procedure allows for mutation from A to a with
0360:      U := u^16) = 2.0E-4.}
0361:
0362: VAR
0363:    Mask   :word;
0364:    rand   :double;
0365:    loc    :word;
0366:
```

```
0367: BEGIN
0368:    rand := random;
0369:    IF rand < 2E-4 THEN                    {mutation ?}
0370:    BEGIN
0371:       loc := random(16)  ;                {where ?}
0372:       IF NOT(odd(Haplo SHR loc)) THEN     {Allele is A ?}
0373:       BEGIN
0374:          Mask := 1 SHL loc;
0375:          Haplo := Haplo OR Mask;     {0 becomes 1, i.e. A becomes a}
0376:       END;
0377:    END;
0378: END;   {of Mutation}
0379:
0380:                  {* * * * * * * * * * * * * * * * *}
0381:
0382: PROCEDURE ConstructZygote(VAR Indi:Individual; ParentPop:Population);
0383:    {This procedure constructs the genotype of a new individual (zygote).}
0384:
0385: BEGIN
0386:    ChooseParent(ParentPop,Parent);
0387:    ExtractHaplotype(Parent.Geno,Haplo);
0388:    Mutation(Haplo);
0389:    Indi.Geno[0] := Haplo;
0390:    ChooseParent(ParentPop,Parent);
0391:    ExtractHaplotype(Parent.Geno,Haplo);
0392:    Mutation(Haplo);
0393:    Indi.Geno[1] := Haplo;
0394: END;   {of ConstructZygote}
0395:
0396:
0397: PROCEDURE Evaluate ( VAR Indi:Individual; VAR survive : boolean);
0398:    {This procedure calculates viability/fertility of an individual
0399:     and tests its survival. Then its genotype is written to the
0400:     genotype-table. It also keeps record of those that don't survive.}
0401:
0402: VAR
0403:    Vint   :real;
0404:  { Fint   :real; }
0405:    lgeno  :ARRAY [1..16] OF byte;         {local genotype of individual}
0406:
0407: BEGIN
0408:    Vint :=1;
0409:  { Fint :=1; }
0410:    FOR l:= 15 DOWNTO 0 DO
0411:    BEGIN
0412:      lgeno[16-l] :=  ((Indi.Geno[0] SHR l) AND 1) +((Indi.Geno[1] SHR l) AND 1);
0413:         Vint := Vint * VMat[16-l,lgeno[16-l]];
0414:       { Fint := Fint * FMat[16-l,lgeno[16-l]];}
0415:    Inc(ActGenoFreq[16-l, lgeno[16-l]]);
0416:    END;
0417:    Indi.Viability := Vint; {Indi.Fertility :=Fint;}
0418:    Inc(actPopsize);
0419:    IF random < Vint THEN survive:=True                {survives?, random is}
0420:                     ELSE survive:=False;              {bad luck number}
0421:    IF survive THEN
0422:    BEGIN
0423:       FOR l:=15 DOWNTO 0 DO  Inc(GenoFreq[16-l, lgeno[16-l]]);
0424:    END;
0425: END;    {of Evaluate}
0426:
0427:                   {* * * * * * * * * * * * * * * *}
```

```
0428:
0429: PROCEDURE CalcQ(VAR u:single;VAR Q:Qarray);
0430:         {This procedure calculates a cumulative probability according to
0431:          Poisson distribution. It is used to determine the number of
0432:          seeds fallen in a certain site.}
0433:
0434: BEGIN
0435:    u := seed*popsize/capacity;         {mean number of seeds per plot}
0436:    Q[0] := exp(-u);
0437:    FOR j := 1 TO 9 DO  Q[j] := u*Q[j-1]/j;
0438:    FOR j := 1 TO 9 DO  Q[j] := Q[j-1] + Q[j];
0439:    Q[10] := 1;
0440: END;
0441:
0442:
0443: PROCEDURE NextGeneration(VAR Genofreq:FreqTable;VAR Pop2:Population);
0444:    {This procedure constructs the next generation by constructing
0445:     new zygotes and testing their survival.}
0446:
0447: VAR
0448:    i,n  : integer;
0449:    rand : single;
0450:
0451: BEGIN
0452:    ClearGenoFreq(GenoFreq);
0453:    ClearGenoFreq(ActGenoFreq);
0454:    ActPopsize := 0;
0455:    CalcQ(u,Q);
0456:    ActViab[gen,sim] := 0;   n:=0;             {n = individual number}
0457:    FOR k := 1 TO capacity DO                      {For each site do}
0458:    BEGIN
0459:       i := 0;
0460:       survive := false;
0461:       rand := random;                   {how many seeds in this site?}
0462:       WHILE (rand > Q[i])  AND (survive = false) DO
0463:       BEGIN                          {as long as seeds left, try again}
0464:          Inc(i);                                {until zygote survives}
0465:          ConstructZygote(Indi,Pop);
0466:          Evaluate(Indi,survive);                          {survive ?}
0467:          ActViab[gen,sim] := ActViab[gen,sim] + Indi.viability;
0468:          IF survive THEN
0469:          BEGIN
0470:             Inc(n);
0471:             Pop2[n] := Indi;
0472:          END;
0473:       END;
0474:    END;
0475:    Popsize := n;                             {size of next generation}
0476:    IF popsize = 0 THEN
0477:    BEGIN
0478:       writeln({#7,}
0479:       'extinction in generation ',gen);
0480:       IF Meanfile THEN
0481:          writeln(outfile,'extinction in generation ',gen);
0482:    END;
0483: END;
0484:
0485:
0486:                {****************File*****************}
0487:
0488:
```

```
0489: PROCEDURE InputToFile(MeanFile:boolean);
0490:   {This procedure writes the input values to the file.}
0491:
0492: VAR
0493:    l : integer;
0494:
0495: BEGIN
0496:   IF MeanFile OR Datafile THEN
0497:   BEGIN
0498:      Writeln(outfile,Filename1);
0499:      Writeln(outfile);
0500:      Writeln(outfile,'number of seeds: ',seed:4:2);
0501:      Writeln(outfile);
0502:      Writeln(outfile,'p0    , fitness matrix');
0503:      FOR l := 1 TO 16 DO
0504:         Writeln(outfile,p0[l]:4:3,' , ',VMat[l,0]:4:3,'   ',VMat[l,1]:4:3,'   ',Vmat[l,2]:4:3);
0505:      Writeln(outfile);
0506:      writeln(outfile,'gen   size   zygote   adult');
0507:   END;
0508: END;   {of InputToFile}
0509:
0510:
0511:
0512: PROCEDURE MeanToStr (VAR MeanStr:str70);
0513:    {This procedure writes the viability-means to a string
0514:     for storing in file.}
0515:
0516: VAR
0517:    st0 ,StGen                        :STRING[3];
0518:    stZMean,stPMean,stInf,stD,stDS    :STRING[6];
0519:    stZS,stPS                         :STRING[6];
0520:
0521: BEGIN
0522:    st0:='   ';
0523:    Str(Gen:3,StGen);
0524:    Str(ActMean[gen]:6:3,StZMean);       Str(ActS[gen]:6:3,stZS);
0525:    Str(PopMean[gen]:6:3,stPMean);       Str(PopS[gen]:6:3,stPS);
0526:    Str(Wtot[gen]:6:3,stInf);
0527:    Str(D[gen]:6:3,stD);                 Str(DS[gen]:6:3,stDS);
0528:    MeanStr := Concat (StGen,st0,stZMean,st0,stZS,st0,stPMean,st0,stPS,
0529:                       st0,stInf,st0,stD,st0,StDS);
0530: END;   {of MeanToStr}
0531:
0532:
0533: PROCEDURE  DepToStr (VAR Depstr:str70);
0534:    {This procedure writes the depression to a string.}
0535:
0536: VAR
0537:   st0 ,stGen          :STRING[3];
0538:   stDep,stDS          :STRING[6];
0539:
0540: BEGIN
0541:    st0 :='   ';
0542:    Str(Gen:3,stGen);
0543:    Str(D[gen]:6:3,stDep);                        {lopend gem. = Dw }
0544:    Str(DS[gen]:6:3,stDS);
0545:    DepStr:= concat(stGen,st0,stDep,st0,stDS);
0546: END;
0547:
0548:
0549:
```

```
0550: PROCEDURE DataToStr(GenoFreq:FreqTable; Pop:population;
0551:                     VAR DataStr:str70);
0552:    {This procedure writes data  of one simulation to a string
0553:     for storing in file.}
0554:
0555: VAR
0556:    st0 ,StGen                :STRING[3];
0557:    stZViab,stPViab           :STRING[6];
0558:    stsize                    :STRING[3];
0559:
0560: BEGIN
0561:    st0:='   ';
0562:    Str(Gen:3,StGen);
0563:    Str(Popsize:3,stsize);
0564:    Str(ActViab[gen,sim]:6:3,StZViab);
0565:    Str(PopViab[gen,sim]:6:3,stPViab);
0566:    DataStr := Concat (StGen,st0,stsize,st0,stZViab,st0,stPViab);
0567: END;    {of DataToStr}
0568:
0569:
0570: PROCEDURE WriteToFile(Datafile:boolean);
0571:    {This procedure writes data of a single simulation to a file.}
0572:
0573: BEGIN
0574:    IF Datafile THEN
0575:    BEGIN
0576:       DataToStr(GenoFreq,pop,datastr);
0577:       writeln(outfile,datastr);
0578:    END;
0579: END;
0580:
0581:                (***************Generation***************)
0582:
0583:
0584: PROCEDURE OneGeneration(VAR Pop,Pop2: Population; VAR GenoFreq: FreqTable);
0585:     {This procedure calculates the data of the present generation,
0586:      creates the next generation and replaces the present population
0587:      by the next.}
0588:
0589: BEGIN
0590:     NextGeneration (GenoFreq,Pop2);
0591:     Pop:= Pop2;                        {replace old by new generation}
0592:     IF popsize > 0 THEN
0593:     BEGIN
0594:        ActViab[gen,sim] := ActViab[gen,sim]/ActPopsize;
0595:        PopViab[gen,sim] := CalcPopViab(Pop);
0596:        Hetero := CalcHetero(GenoFreq);
0597:        PopOutPut(Pop,ActGenoFreq,GenoFreq);
0598:        WriteToFile(datafile);
0599:     END;
0600: END;           {of OneGeneration}
0601:
0602:
0603:                {*************Infinite*************}
0604:
0605:
0606: PROCEDURE Infinite(VMat:Matrix: VAR Wtot:MeanArray);
0607:     {This procedure calculates the viability of an infinite population
0608:      for the same selection and initial frequencies as Pop.}
0609:
0610: VAR
```

```
0611:    Wint    :real;
0612:    W       :ARRAY [1..16] OF single;
0613:
0614: BEGIN
0615:    InitFreq(pa2);
0616:    FOR gen := 0 TO generation DO
0617:    BEGIN
0618:       Wint := 1;
0619:       FOR l := 1 TO 16 DO
0620:       BEGIN
0621:          pa[l] := pa2[l];
0622:          W[l] := (SQR(1-pa[l]) * VMat[1,0]) + (2 * pa[l] * (1-pa[l]) * VMat[1,1])
0623:                  + (SQR(pa[l]) * VMat[1,2]);
0624:          Wint := Wint * W[l];
0625:          pa2[l] := pa[l] * (pa[l] * VMat[1,2] + (1-pa[l]) * VMat[1,1])/W[l];
0626:       END;
0627:       Wtot[gen] := Wint;
0628:    END;
0629: END;    {of Infinite}
0630:
0631:               {******************Over All******************}
0632:
0633:
0634: PROCEDURE CalcMean(Viab:TotArray; VAR Mean,S:MeanArray);
0635:    {This procedure calculates the arithmic mean and standard deviation
0636:     of a given parameter over the simulations.}
0637:
0638: VAR
0639:    Int,Varint : real;
0640:    simu       : integer;
0641:
0642: BEGIN
0643:    FOR gen := 0 TO generation DO
0644:    BEGIN
0645:       Int := 0;
0646:       Varint := 0;
0647:       FOR simu := 1 TO simulation DO                          {mean}
0648:          Int := Int + Viab[gen,simu];
0649:       Mean[gen] := Int / simulation;
0650:       FOR simu := 1 TO simulation DO                          {st.deviation}
0651:          Varint := Varint + SQR(Viab[gen,simu] - Mean[gen]);
0652:       S[gen] := SQRT Varint/simulation);
0653:    END;
0654: END;    {of CalcMean}
0655:
0656:
0657: PROCEDURE  MeanAll;
0658:    {This procedure calculates the mean and standard deviation of
0659:     two viability measures and the viability of an infinite population
0660:     and sends them to file.}
0661:
0662: BEGIN
0663:    CalcMean(PopViab,PopMean,PopS);
0664:    CalcMean(ActViab,ActMean,ActS);
0665:    Infinite(VMat,Wtot);
0666: END;    {of MeanAll}
0667:
0668:
0669: PROCEDURE CalcDepression(VAR D,Dw:MeanArray);
0670:    {This procedure calculates the fitability of the finite population
0671:     (Pop) relative to the infinite population.}
```

```
0672:
0673: VAR
0674:    a,b,c   :integer;
0675:    Dint    :real;
0676:    Dt      :ARRAY [9..generation] OF real;
0677:
0678: BEGIN
0679:    MeanAll;
0680:    FOR gen := 0 TO generation DO
0681:    BEGIN
0682:        D[gen]  := 1 - ActMean[gen]/Wtot[gen];
0683:        DS[gen] := ActS[gen]/Wtot[gen];
0684:    END;
0685:
0686: (*   If Depfile then
0687:    begin
0688:      Dint := 0;                              {walking mean over 10 generations}
0689:      FOR a := 5 TO 14 DO                      {starting at generation 5}
0690:      Dint := Dint + D[a];
0691:      Dt[14] := Dint;
0692:      FOR b := 14 TO (generation-1) DO
0693:      Dt[b+1] := Dt[b] + D[b+1] - D[b-9];
0694:      FOR c := 0 TO 14 DO
0695:      Dw[c] := Dt[14]/10;
0696:      FOR c := 15 TO generation DO
0697:      Dw[c] := Dt[c]/10;
0698:    END;*)
0699:
0700:    FOR gen := 0 TO generation DO          {write to file}
0701:    BEGIN
0702:      IF DepFile THEN
0703:      BEGIN
0704:         DepToStr(DepStr);
0705:         Writeln(outfile, Depstr);
0706:      END;
0707:   END;
0708: END;       {of CalcDepression}
0709:
0710:
0711: PROCEDURE EndFix (GenFreq:FreqTable);
0712:      {This procedure writes fixations in last generation to file.}
0713:
0714: BEGIN
0715:    IF MeanFile THEN
0716:    BEGIN
0717:       Write(outfile,'simulation ',sim,'   '):
0718:       FOR l := 1 TO 16 DO                                {check fixation}
0719:       BEGIN
0720:          IF (GenoFreq[l,0] = popsize) THEN
0721:             Write(outfile,'-  ') ELSE                      {A fixed : -}
0722:          IF (GenoFreq[l,2] = popsize) THEN
0723:             Write(outfile,'+  ') ELSE                      {a fixed : +}
0724:             Write(outfile,'0  ');
0725:       END:
0726:       Writeln(outfile):
0727:    END;
0728: END;    {of EndFix}
0729:
0730:
0731:               {* * * * * * * *Compare* * * * * * * *}
0732:
```

```
0733:
0734: PROCEDURE AllOutput(MeanFile:boolean);
0735:     {This procedure viability means and st.deviations to a file.}
0736:
0737: BEGIN
0738:    IF MeanFile THEN
0739:    BEGIN
0740:       Writeln(outfile);
0741:       Writeln(outfile,' gen.  ZygoteV    s     adultV     s       infV     Depinf    s');
0742:       FOR gen := 0 TO 9 DO
0743:       BEGIN
0744:          MeanToStr(MeanStr);
0745:          Writeln(outfile,MeanStr);
0746:       END;
0747:       gen := 10;
0748:       REPEAT
0749:          MeanToStr(MeanStr);
0750:          Writeln(outfile,MeanStr);
0751:          gen:= gen + 5;
0752:       UNTIL gen > generation;
0753:     END;
0754: END;
0755:
0756: {***************************** Main Program ******************************}
0757:
0758: BEGIN
0759:    FOR x := 1 TO 1 DO
0760:    BEGIN
0761:        OpenFile;
0762:        Randomize;
0763:        InitFreq(p0);
0764:        InitVMatrix;
0765:        InputToFile(MeanFile);
0766:        sim := 1;
0767:        WHILE sim <= simulation DO
0768:        BEGIN
0769:           ClrScr;
0770:           Writeln('SIMULATION ',sim);
0771:           InitGeneration(p0,Pop,GenoFreq);          {Create generation 0}
0772:           gen := 0;
0773:           Pop0OutPut(Pop,GenoFreq);
0774:           WHILE (gen < generation)  AND (popsize>0) DO
0775:           BEGIN
0776:             gen := gen + 1;                          {replace population}
0777:             OneGeneration(Pop,Pop2,GenoFreq);        {and evaluate      }
0778:           END;
0779:           IF (popsize=0) AND(gen<generation) THEN
0780:           REPEAT
0781:               gen := gen + 1;
0782:               ActViab[gen,sim] := 0;            {extinct populations have}
0783:               PopViab[gen,sim] := 0;            {fitness = 0}
0784:           UNTIL gen = generation;
0785:           EndFix(GenoFreq);
0786:           Delay(1000);                                    {wait 1 sec}
0787:           sim:= sim+1;
0788:        END;
0789:        CalcDepression(D,Dw);
0790:        AllOutput(MeanFile);
0791:        Close(outfile);
0792:    END;
0793:    REPEAT UNTIL KeyPressed;
```

```
0794: END.
0795:
0796: (************************************END*********************************)
0797:
0798:
0799:
```