UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Mathematical modelling and parentage analysis of Hybrid Zones

Ferran Palero Pastor

INVESTMAT MSc Thesis

Valencia, December 2009

Departament de Matemàtica Aplicada

ABSTRACT

Hybrid zones are narrow regions in which species exchange genes, and yet remain distinct. This situation represents a clash between two views of species; one based on the pattern of gene flow, and the other on the maintenance of a cluster of phenotypes that is stable to invasion by foreign genes. The present study aims to 1) review the theoretical framework behind the mathematical analysis of clines in nature; 2) Implement a simplified version of these theoretical models in *Mathematica*; 3) Design an 'in silico' setting to simulate reproduction while keeping track of the resulting pedigrees; and 4) test the efficiency of parentage analysis methods implemented in R and estimate a minimum number of molecular markers needed to reconstruct parentage in hybrid zones. Our results indicate that, for a given level of migration between demes, different levels of selection have a direct impact on the cline width, with stronger natural selection causing clines to be narrower. On the other hand, selection and recombination also seem to play a significant role on the distribution of the level of linkage disequilibrium along the cline. Finally, it was found that a minimum number of 18 molecular markers (with at least 5 alleles each) will be needed in order to obtain a 95% confidence on our parentage assignments.

RESUM

Les zones híbrides són regions estretes en les quals troben espècies que intercanvien gens, i tanmateix romanen diferenciades. Aquesta situació representa un enfrontament entre dues visions d'espècie; una basada en el patró de flux gènic, i l'altra en el manteniment d'un grup d'individus amb un fenotip estable. Aquest treball pretén 1) revisar la teoria en què es basa l'anàlisi matemàtica de clines; 2) implementar una versió simplificada d'aquests models teòrics a Mathematica; 3) Establir un disseny 'in silico' que permeta simular reproducció biològica i alhora registre els pedigrís resultants; i 4) provar el rendiment dels mètodes d'anàlisi de parentiu implementats a R i calcular un nombre mínim de marcadors moleculars necessaris per reconstruir pedigrís en zones híbrides. Els nostres resultats indiquen que, per a un nivell donat de migració entre localitats, els diferents nivells de selecció tenen un impacte directe en l'amplada de la clina, amb una selecció major resultant en clines més estretes. D'altra banda, selecció i recombinació també semblen jugar un paper significatiu en la distribució del nivell de desequilibri de lligament al llarg de la clina. Finalment, es troba que un mínim de 18 marcadors moleculars (amb com >5 al·lels cadascun) són necessaris per obtenir un 95% de confiança en les nostres assignacions de parentiu.

CONTENTS

Mathematical modelling and Parentage analysis of Hybrid Zones

Ferran Palero

Institute of Science and Technology Austria (ISTA)
Am Campus 1
A – 3400 Klosterneuburg
Austria.
e-mail: fpalero@ist.ac.at

"It is not clear how Templeton's cohesion species differs from a biological species, although I suspect a cohesion species would possess additional reproductive barriers acquired during the years of deliberation on its status."
(Schemske 2000)

Abstract

A biological species is defined as an "actually or potentially interbreeding group of populations". In practice, any taxa that can produce viable and fertile F2 generations are regarded as being of the same species. However, hybrid zones are narrow regions in which genetically distinct populations meet, mate and produce hybrids. Therefore, species exchange genes, and yet remain distinct. This situation represents a clash between two views of species; one based on the pattern of gene flow, and the other on the maintenance of a cluster of phenotypes that is stable to invasion by foreign genes. In order to explain how selection can maintain well-defined species despite gene flow, a diffusion equation approximation to hybrid zones is presented in this report. Besides representing the basis for a challenging theoretical model on gene flow-selection equilibrium, stable clines present an interesting situation for the analysis of paternity and pedigree reconstruction methods.

The present study aims to 1) shortly review the theoretical framework behind the mathematical analysis of clines in nature, pointing out the most relevant assumptions made by the models; 2) Implement a simplified version of these theoretical models in *Mathematica*, in order to describe the quantitative effect of different levels of dispersal and selection on the shape of the cline; 3) Design an 'in silico' setting to simulate reproduction while keeping track of the resulting pedigrees; and 4) test the efficiency of parentage analysis methods implemented in R and estimate a minimum number of molecular markers needed to reconstruct parentage in hybrid zones.

Our results indicate that, for a given level of migration between demes, different levels of selection have a direct impact on the cline width, with stronger natural selection causing clines to be narrower. On the other hand, selection and recombination also seem to play a significant role on the distribution of the level of linkage disequilibrium along the cline. Finally, our simulations indicate that a minimum number of 18 molecular markers (with at least 5 alleles each) will be needed in order to obtain a 95% confidence on our parentage assignments. All the analyses described in this paper were performed using a Mathematica 7.0 notebook.

Introduction

Hybrid Zones : Natural Selection in action

Hybrid zones are examples of stepped clines, which are dramatic geographic gradients in the frequency of a gene or a trait. One might imagine that hybrid zones are rather rare and hard to find. However, we have many examples of clines in nature: differences in skin color in humans, differences in color pattern in the European fire-bellied toad (*Bombina bombina*) and the yellow-bellied toad (*Bombina variegata*), differences in chromosome number in *Mus musculus*, and many others (Barton and Hewitt, 1985; Capanna, 1980). In fact, the present study is originally motivated by a striking *Antirrhinum* hybrid zone located in the Catalonian Pyrenees (Whibley et al., 2006). The snapdragon *Antirrhinum* shows two different flower colors in a very narrow area, less than 10km wide, near Planoles. As you travel along the main road, flowers shift from being pink on one extreme to yellow in the other, and only a few hybrids (orange flowers) can be observed in between. In order to explain the scarcity of hybrids in this area, it has been hypothesized that orange flowers (hybrids) are selected against by pollinators (Bumblebees: *Bombus terrestris*), which will make them less fertile and impede their expansion. However, several questions arise from these considerations: how strong should be natural selection against hybrids to maintain the observed differences? To which extent does pollen dispersal (gene flow) between pink and yellow flowers influence the shape of the cline?

A biological species is defined as an "actually or potentially interbreeding group of populations". In principle, we would expect gene flow and dispersal to homogenize the genetic composition of the populations in hybrid zones, but instead they remain distinct. How could it be? This represents a clash between two views of species; one based on the pattern of gene flow, and the other on the maintenance of a cluster of phenotypes that is stable to invasion by foreign genes. Therefore, hybrid zones offer us several ways of understanding the nature and origin of species. The wide range of genotypes found in a hybrid zone can be used to analyze the genetic differences and selective forces that separate the taxa involved. This may allow some inferences about the way these differences evolved and, by extrapolation, about the way fully isolated species diverge from each other. As will be shown in the present report, studies of hybrid zones allow us to quantify the genetic differences responsible for speciation, to measure the diffusion of genes between diverging taxa, and to understand the spread of alternative adaptations.

A more practical reason for developing a model of gene flow and selection is to increase our understanding of the causes of the observed spatial patterns of gene frequencies. In fact, Haldane (1948) original work on this subject (see below) was motivated by the problem of measuring selection in *Mus musculus* and has been used by others to estimate the strength of selection in other natural population. Despite being a key concept in the Darwinian theory of evolution, estimating selection in natural conditions remains a challenging task. One of the most accessible ways to estimate the strength of selection in nature is to measure the rate of change in gene frequencies in a cline and compare with those results expected under a particular model. That is still another reason why studying hybrid zones should be a priority for evolutionary biologists.

Overview : Models of clines in continuous habitats

Several models have been proposed in order to account for the existence of clines in natural populations. The available models of clines in continuous habitats can be arranged into two classes. In the first class, dispersal is negligible. Selection maintains a stable equilibrium at each locality (e.g. through heterozygote advantage). In that case, the cline just mirrors a smooth gradient in selection coefficients and hence in the equilibrium point. We will call these **dispersal-independent clines**; and they include Moore's (1977) "bounded hybrid superiority". In the second class, the homogenizing effect of dispersal is balanced against some cause of spatial heterogeneity. Most theoretical work has been on such models (Felsenstein, 1976). They include neutral clines, in which an initially steep gradient decays with time; waves of advance of an advantageous allele (Fisher, 1937); and dispersal-selection balance, in which either differences in environment (Haldane, 1948) or selection against intermediate genotypes (heterozygotes or recombinants) (Bazykin, 1969, 1972) maintains a stable cline. We will refer to the last type as a **tension (hybrid) zone**.

The distinction between these two classes depends on the characteristic **scale of selection**, l, where $l \equiv \frac{\sigma}{\sqrt{s}}$, with σ^2 = dispersal rate (more precisely, the variance in distance between parent and offspring). The selection coefficient (s) is proportional to selection or, for a neutral cline, it is the inverse of the time since contact (Slatkin, 1973). Any **dispersal-**

dependent cline has a width (w, defined as the inverse of the maximum gradient) of the same order as l. Conversely, if selection is to maintain a **dispersal independent cline**, w must be much greater than l. A cline can still be regarded as a dispersal-selection balance even if some hybrid genotypes are favored, provided they are only favored within a region much narrower than l. When many clines coincide, linkage disequilibria will be generated by the dispersal of parental combinations of alleles into the center (Slatkin, 1975). If many genes are involved and selection is comparable with recombination, disequilibria will induce a sharp step in each cline, flanked by long tails of introgression (see below). The central region of the cline (in which disequilibria are strong) will be distinct from the surrounding tails, and it has been proposed that its width will depend strongly on the ratio between selection and recombination (Barton, 1983).

Simulating hybrid zone evolution "in silico" and Parental assignment

In 1991, Barton and Turelli developed recursions to describe the evolution of multilocus systems under arbitrary forms of selection. Later, Kirkpatrick et al. (2002) generalized their approach to allow for arbitrary modes of inheritance, including diploidy, polyploidy, sex linkage, cytoplasmic inheritance, and genomic imprinting. The framework was also extended to allow for other deterministic evolutionary forces, including migration and mutation. Exact recursions that fully describe the state of the population were presented and implemented in a computer algebra package called MULTILOCUS. The present study builds up on that work, by extending their approach to the analysis of hybrid zones. Furthermore, since we are particularly interested in analysing real data from the *Antirrhinum* hybrid zone, our study will include further simulations on pedigree construction, therefore increasing the current functionality of the package.

As previously pointed out, selection and dispersal are key factors determining the main traits of the hybrid zone, such as cline shape and width. Therefore, in order to estimate dispersal rate (or the variance in distance between parent and offspring) and selection (or reproductive success), we should be able to trace the evolution of a biological system from generation to generation, describing which individuals are able to produce viable offspring and how far from their parents do these descendants get established. This is not an easy task, since tagging and tracking every individual in a population would be prohibitive. Nevertheless, recent studies indicate that polymorphic genetic markers are potentially helpful in resolving genealogical relationships among individuals in a natural population (Jones and Ardren, 2003). Our study will further investigate which is the minimum number of molecular markers needed in order to have high confidence on our paternity estimates.

Aims

In summary, the present study aims to 1) shortly review the theoretical framework behind the mathematical analysis of clines in nature, pointing out the most relevant assumptions made by the models; 2) Implement a simplified version of these theoretical models in *Mathematica*, in order to describe the quantitative effect of different levels of dispersal and selection on the shape of the cline; 3) Design an 'in silico' setting to simulate reproduction while keeping track of the resulting pedigrees; and 4) test the efficiency of parentage analysis methods in reconstructing the simulated pedigrees.

Methods: Theoretical Framework

Modelling Hybrid Zones

Quick introduction to genetics jargon

We now give a brief explanation of some of the biological and genetical terms used. These definitions may not be entirely comprehensive, however they are adequate for the purposes of this thesis. Unless stated otherwise, we consider a **diploid** population, so that each individual possesses two sets of chromosomes, one set inherited from each parent. We are only interested in the genes located at one particular **locus** (i.e. the genes at a particular place on a particular chromosome). We consider the situation in which the gene occurs in two different forms, called **alleles**. The Mendelian model of inheritance assumes that parents pass on discrete heritable units - genes - that remain separate and can be passed on to subsequent generations in undiluted form.

The genetic makeup of an individual is known as its **genotype**. A **homozygote** possesses two identical alleles for a given trait, whereas a heterozygote has two different alleles for a given trait. The physical traits exhibited by an individual is known as its **phenotype**. In a heterozygote, the allele that is fully expressed by the phenotype is known as the dominant allele, whereas if an allele is completely masked in the phenotype, it is known as **recessive**. A change in

the gene frequencies in a population is the most elementary step in evolution from a population genetics point of view. Variation exists in a population due to the different possible alleles an individual may possess, and this variation is heritable. According to the Darwinian theory of natural selection, individuals with variations that are best suited to the environment are more likely to survive and pass on their advantageous genes to successive generations.
Biologists hope to determine the probability of the ultimate success of an advantageous gene by attempting to model these changes in allele frequencies. In order to do so, population genetics uses some deterministic models that have great similarities with those used in other disciplines. In this thesis we will focus particuarly on diffusion equations applied to the analysis of hybrid zones.

The Diffusion equation in population genetics : "Fisher's Wave of Advance".

The origins of the application of a diffusion approximation in population genetics are to be found in the pioneering work of Fisher (1930, 1937). He considered the case of a population distributed in a linear habitat, such as a shoreline, which is occupied with uniform density. If at any point of the habitat an advantageous mutation occurred, we would expect the mutant gene to increase at the expense of the alleles previously occupying the space around the new mutant. This process will later, as the advantageous gene is diffused into the surrounding population, expand in the adjacent portions of its range. Supposing the range to be long compared with the distances separating the sites of offspring from those of their parents, there will be, advancing from the origin, a wave of increase in the gene frequency.

Let p be the **frequency of the mutant gene**, and q that of its parent allele, which we shall suppose to be the only other allele present. Let s be the intensity of selection in favour of the mutant gene, supposed independent of p. If we further suppose that the **rate of diffusion per generation** across any boundary may be equated to

$$-k\frac{\partial p}{\partial x} \tag{1}$$

at that boundary, x being the coordinate measuring position in the linear habitat, then the allele frequency of the mutant allele p must satisfy the differential equation:

$$\frac{\partial p}{\partial t} = k\frac{\partial^2 p}{\partial x^2} + spq \tag{2}$$

where t stands for time in generations.

Notice that we are stating that the rate of change in allele frequency through time equals the sum of the allele frequency change through space due to diffusion plus the amount due to selection. This will become important later.

The constant k is a **coefficient of diffusion** analogous to that used in physics. Its use should be appropriate in many cases. Of course, in all real cases we may expect irregularities due to k varying at different points of the range, due to variations in the density of the population, and to variation in the selective advantage of the mutant at different places. Further, the means of diffusion may involve an unequal drift in opposite directions (anisotropy), so that some parts of the range predominate as centres of production and others as centres of extinction. Nevertheless, the purpose of equation (1) is to specify the simplest possible conditions.

If we seek for a solution representing a **wave of stationary form** advancing with velocity v, we may put

$$\frac{\partial p}{\partial t} = -v\frac{\partial p}{\partial x} \tag{3}$$

and obtain the differential equation (2) involving only one independent variable:

$$k\frac{d^2 p}{dx^2} + v\frac{dp}{dx} + spq = 0 \tag{4}$$

Since the variable x does not appear explicitly, we may define, for the frequency gradient,

$$g = -\frac{dp}{dx} \tag{5}$$

which allows us to write

$$\frac{d^2 p}{dx^2} = -\frac{dg}{dx} = g\frac{dg}{dp} \tag{6}$$

and therefore find the relation between the frequency gradient (g) and the mutant allele frequency (p),

$$k\frac{dg}{dp} - vg + spq = 0 \tag{7}$$

At the point of inflexion, we will have $\frac{dg}{dp} = 0$, and vg = spq; in advance of this point $\frac{dg}{dp} > 0$.

If the ratio between the frequency gradient and the mutant allele frequency tends to a limit value when p tends to zero

$$\lim_{p\to 0} \frac{g}{p} = u \tag{8}$$

then u must satisfy the equation

$$ku^2 - vu + s = 0 \tag{9}$$

which is a quadratic equation in u that has real roots only if v^2 is not less than 4ks. Notice that $\frac{g}{p}$ cannot tend to zero for vg > spq, and cannot tend to infinity because $v > k\frac{dg}{dp}$. Therefore, solutions only exist for which the velocity of propagation is equal to, or exceeds, $2\sqrt{ks}$.

The most striking point about equation (4) is that the velocity of advance of the mutant factor appears to be indeterminate. If, for example, any part of the range were filled with the mutant form, and the zone of transition were artificially given frequencies with the low gradient of gene ratio appropriate to a high velocity, the mutation would spread with a higher velocity than if the initial gradient had been higher, and would continue to spread indefinitely with this higher velocity so long as uniform conditions were encountered.

Ultimately, the velocity of advance would adjust itself so as to be the same irrespective of the initial conditions. If this is so, equation (4) must omit some essential element of the problem, and it is indeed clear that while a coefficient of diffusion may represent the biological conditions adequately in places where large numbers of individuals of both types are available, it cannot do so at the extreme front and back of the advancing wave, where the numbers of the mutant and the parent gene respectively are small, and where their distribution must be largely sporadic. This reasoning indicates that the diffusion approximation would not be aplicable under boundary conditions.

Fisher further defined the effect of chance at the advancing front, which he calculated by considering an aggregate of discrete particles, which increase in number with a relative growth rate s, as at the wave front of our original problem, but are free also to increase in numbers indefinitely in the interior of their range. We shall suppose them to be scattered at small unit intervals of time so that the displacement of the particles at each scattering are is independent and normally distributed

$$N(x, \sigma) \sim \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\frac{x^2}{\sigma^2}} dx \tag{10}$$

where k of our previous notation will correspond to $\frac{\sigma^2}{2}$.

Finally, the indeterminacy of velocity can be resolved by comparison with the properties of multiplying aggregates of particles, constantly subjected to random scattering. It appears that the actual velocity of advance must be the minimum compatible with the differential equation. This velocity is proportional to the square root of the intensity of selective advantage and to the standard deviation of scattering in each generation, or to the square root of the diffusion coefficient when time is measured in generations.

It may be expressed in the form

$$v = \sigma\sqrt{2s} = 2\sqrt{ks} \tag{11}$$

where, again, s is the selectivc advantage, σ the standard deviation of scattering, and k the diffusion coefficient.

The "length" of the wave, or the distance between any two assigned gene ratios, is proportional to $\lambda = \sqrt{\frac{k}{s}}$, wich may be taken as the unit of length.

In summary, Fisher considered a favorable mutant arising in a continuously distributed population and presented a partial differential equation for the deterministic change in the geographic pattern of gene frequencies. Equation (2) predicts a wave front of rising gene frequencies propagating through the population and leaving behind it a region fixed for the favored allele. Since the allele was assumed to be favorable in all parts of the population, no stable cline is achieved. However, it has been shown that the velocity of advance of the wave will be proportional to $\sqrt{s}$, and the length of the wave front to $\frac{\sigma}{\sqrt{s}}$. Despite there is some indeterminacy in the shape and speed of the wave, which can depend on the initial conditions, Fisher argued that his solution, would be the ultimate result after initial condition effects were lost.

Mathematical Model of a Hybrid Zone : Theory of a cline

Fisher' s Wave of Advance theoretical framework formed the basis for subsequent modelling of clines (Haldane, 1948) and hybrid zones (Bazykin, 1969). Equation (2) predicts a wave front of increasing allele frequency, propagating through the population. Only original alleles are present in front of the wave, and behind the wave is an area taken over by the mutant allele. However, when dealing with clines, one needs to consider that local adaptation might make one allele (let's say A) favourable in part of the environment while the alternative allele (let's say a) could be favoured in the other area. Therefore, we are not interested in the propagation of a new mutation along a linear habitat, but rather to the conditions that make selection and dispersal to remain in stable equilibrium.

Assuming that the population is in Hardy-Weinberg equilibrium and if the frequency of the gene a in adults at a distance x from the boundary is p, then the frequency of the recessive phenotype aa will be $z = p^2$. When p is plotted against x we get a sigmoideal curve (Fig. 1), with $p \to 0$ as $x \to -\infty$, and $p \to 1$ as $x \to \infty$.

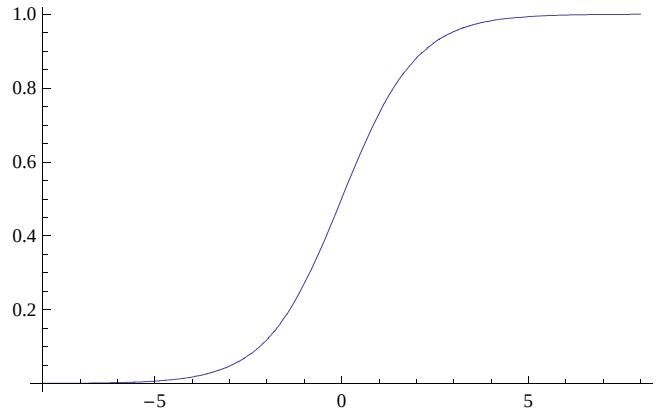

Figure 1. Standard logistic sigmoid function

When x = 0, i.e. on the boundary between the two species, dp/dx is continuous, since any discontinuity would be smoothed out by migration, but note that $\frac{\partial^2 p}{\partial x^2}$ changes sign abruptly. Note that the fitnesses of the genotypes change sharply at the border between the regions. In the half plane where x is positive, we are assuming that aa zygotes have a fitness 1 + s times that of AA and Aa, and therefore they will become more frequent there. In the other half plane the fitness is 1 - s, with the selection coefficient being small and positive ($0 < s << 1$).

We are also assuming that individuals migrate at random. A group of individuals born at distance x breed at distances x + t, where t is normally distributed about zero with unit standard deviation. That is, we take as our unit of distance the root of the mean square of the distances travelled by an animal between birth and breeding in the direction normal to the boundary. In fact, since migration does not depend on genotypes, we can consider the genes, not the zygotes, as migrating.

Now let's f(t) be the frequency distribution function of t, symmetrical about zero. At the point x + t the gene frequency is obtained from the Taylor Series Expansion of the function:

$$f(x_0 + t) = f(x_0) + \sum_{n=1}^{\infty} \frac{t^n}{n!}\left[\frac{d^n f}{dx^n}\right]_{x=x_0} \tag{12}$$

Note that f() = p.

As a result of 1 year's migration, the frequency at x changes from p to:

$$\int_{-\infty}^{\infty} p\, f(t)\, dt = \tag{13}$$

$$p + \frac{1}{2!}\frac{d^2 p}{dx^2}\int_{-\infty}^{\infty} t^2 f(t)\, dt + \frac{1}{4!}\frac{d^4 p}{dx^4}\int_{-\infty}^{\infty} t^4 f(t)\, dt + \ldots = p + \frac{\overline{t^2}}{2!}\frac{d^2 p}{dx^2} + \frac{\overline{t^4}}{4!}\frac{d^4 p}{dx^4} + \ldots$$

And for a normal distribution this is

$$p + \frac{\sigma^2}{2!}\frac{d^2 p}{dx^2} + \frac{3\sigma^4}{4!}\frac{d^4 p}{dx^4} + \frac{15\sigma^6}{6!}\frac{d^6 p}{dx^6} + \ldots \tag{14}$$

Provided that the selection coefficient is sufficiently small and the distribution is not too leptokurtic, we can neglect terms after the second, and since we assume $\sigma^2 = 1$, we have the familiar diffusion expression:

$$f(x + t) = 1 + \frac{1}{2}\frac{d^2 p}{dx^2} \tag{15}$$

As a result of selection the ratios of the genotypes are altered, when x > 0, from:

$$(1 - p)^2 \sim AA : 2p(1 - p) \sim Aa : p^2 \sim aa$$

to

$$(1 - p)^2 \sim AA : 2p(1 - p)^2 \sim Aa : (1 + s)p^2 \sim aa$$

Thus the frequency of a is altered from p to $\frac{p+sp^2}{1+sp^2}$, or approximately to $p + sp^2(1 - p)$. Since selection and migration (gene flow) are in equilibrium, we get the following equations for both sides of the cline:

$$\frac{d^2 p}{dx^2} = -2sp^2(1 - p) \tag{16}$$

$$\frac{d^2 p}{dx^2} = sp^2(1 - p) \tag{17}$$

To solve this pair of differential equations, we can do as before and replace $g = \frac{dp}{dx}$. Then, for x > 0,

$$g\frac{dg}{dp} = -2sp^2(1 - p) \tag{18}$$

so

$$\frac{g^2}{2} = \int g\, dg = -2s\int p^2(1 - p)\, dp = C - 2s\left(\frac{p^3}{3} - \frac{p^4}{4}\right) \tag{19}$$

Given that in the limit when $x \to \infty$, $p \to 1$ and $g \to 0$, we will have that $C = \frac{s}{6}$ and

$$g^2 = \left(\frac{dp}{dx}\right)^2 = \frac{s}{3}\left(1 - 4p^3 + 3p^4\right) = \frac{s}{3}(1 - p)^2\left(1 + 2p + 3p^2\right). \tag{20}$$

Similarly, when $x < 0$, $g^2 = \left(\frac{dp}{dx}\right)^2 = 2C' + 4s\left(\frac{p^3}{3} - \frac{p^4}{4}\right)$. So when $x \to -\infty$, $p \to 0$ and $g \to 0$, we have $C' = 0$ and

$$g^2 = \left(\frac{dp}{dx}\right)^2 = \frac{sp^3}{3}(4 - 3p) \tag{21}$$

Now, in the middle of the cline, when $x = 0$, $p = b$ and $\frac{dp}{dx}$ has the same value for both branches of the cline. Hence we get $\frac{s}{3}\left(1 - 4b^3 + 3b^4\right) = \frac{s}{3}\left(4b^3 - 3b^4\right)$, or simplifying $3b^4 - 4b^3 + \frac{1}{2} = 0$. This equation, giving the value of the allele frequency on the middle of the cline, has one and only one root between 0 and 1, which can be found by iteration.

Mathematical Model of a Hybrid Zone : Moving into 2-dimensions

Bazykin (1969) suggested the **simplest model of a tension zone**, which is similar to the one that we will follow in our simulations. In this case, we assume that heterozygotes have fitness 1 - s relative to either homozygote. If the allele frequency is p, the change in allele frequency is then (for small s):

$$\frac{\partial p}{\partial t} = \frac{\sigma^2}{2}\frac{\partial^2 p}{\partial x^2} + spq(p - q) \tag{22}$$

where $p + q = 1$.

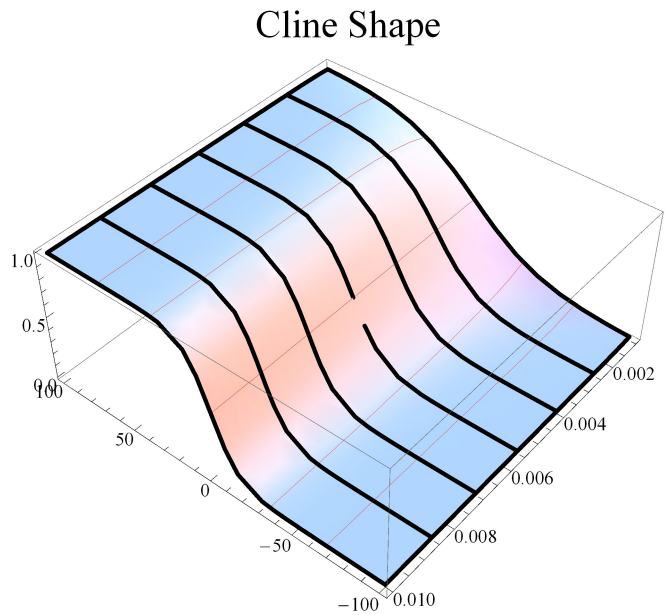

Figure 2. Change in allele frequency along the cline ($-100 < x < 100$) at different selection levels ($0.001 < s < 0.01$).

As stated before, this equation describes the situation in which "the heterozygote is less fit than both homozygotes" and "homozygote fitness is equal". Again, remember that the dispersal rate σ is defined as the standard deviation of the distance between parent and offspring along a chosen axis and it has units of $(\text{distance} * \text{time})^{-1/2}$.

Bazykin' s model is based on a special kind of **reaction-diffusion equations,** in which attention is centred not on the spontaneous formation of a pattern from a homogeneous field, but on the interaction between areas which have moved to different states. This particular equation has the solution:

$$p = \frac{1}{1 + e^{-\frac{x - x_0}{l}}} \tag{23}$$

where $l = \sqrt{\frac{\sigma^2}{2s}}$ is the characteristic distance over which selection changes allele frequencies (Fig. 3),

and the cline width is 4 l.

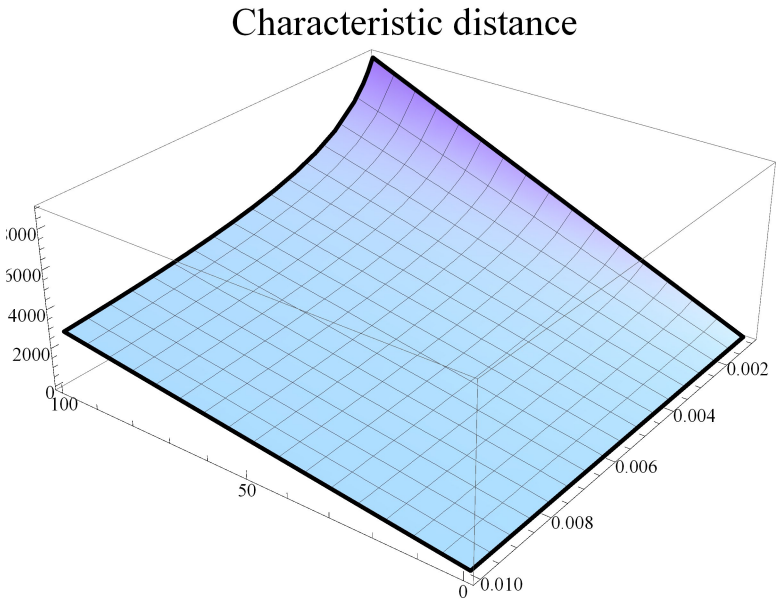

Figure 3. Change in characteristic length with dispersal ($0 < \sigma < 100$) and selection ($0.001 < s < 0.01$).

In this model, the environment is homogeneous and the homozygotes have equal fitnesses, so the cline can form anywhere (x_0), and the equation has a family of neutrally stable solutions that are known as '**topological solitons**'.

Topological solitons are solitary waves set up at the boundary between two regions constrained to be in different states. The same phenomenon (often the same equation) arises in many other fields. Topological solitons are solutions of systems of partial differential equations representing stable structures which are localized in space, with their stability being due (in part) to non-trivial topology. Systems admitting such solitons have been known for a long time; the first examples were in one space dimension, whereas more recent work has concentrated on structures in two and three space dimensions. Investigation of the mathematical properties of topological solitons has proceeded alongside the search for their applications in the description of numerous processes and phenomena in physics and biology. Nevertheless, their application to population genetics is very recent and mostly unexplored.

Statistical theory of paternity inference

Now we move to a completely different topic, but that will be needed in order to get estimates of dispersal to be used in our hybrid zone model. Parentage analysis is a precise form of assignment testing that can be particularly useful for detecting ecological and evolutionary patterns in systems with high levels of gene flow (Manel et al., 2005). Such systems have limited genetic differentiation, which severely restricts the utility of population-level assignment methods. Therefore, parentage analyses may allow for the inference of gene flow and dispersal at ecologically relevant timescales.

A challenge to employing parentage analysis in natural populations is that large population sizes, variable dispersal distances and high rates of mortality may severely constrain the number of sampled parent-offspring pairs. These challenges are amplified in systems where patterns of dispersal are unobservable, such as when propagules are too small to track directly (e.g. pollen dispersal). In addition, because of a lack of pragmatic methods, long-distance dispersal events are often ignored or remain undetected in many species of plants (Nathan, 2006), fungi (Kauserud et al., 2006) and animals that are cryptic or have complex life histories (Derycke et al., 2008). Nevertheless, large genotypic data sets

may still be used to uncover some of these enigmatic processes, and parentage analysis can be a powerful tool for the direct detection of patterns of dispersal and population connectivity. As shown in the previous section, dispersal plays a key role in defining cline behaviour and estimation of selection. Therefore, being able to assign parentage is critical when dealing with hybrid zones.

Methods of parentage analysis

Several methods of parentage analysis are available in the literature. Exclusion is based on Mendelian rules of inheritance and uses incompatibilities between parents and offspring to reject particular parent-offspring hypotheses. Categorical and fractional likelihood assign progeny to non-excluded parents based on likelihood scores derived from their genotypes. The categorical technique assigns the entire offspring to a particular male, whereas the fractional technique splits an offspring among all compatible males.

These methods are expected to perform differently from one another when estimating the variances in reproductive or mating success for one or both sexes in a population. Thus, the categorical method overestimates the reproductive success of individuals with many homozygous loci, the fractional technique requires the researcher to set a prior probability of parentage, etc. Moreover, although they might seem straightforward in principle, such analyses are usually complicated by several factors such as incomplete sampling of potential parents, individuals being related to each other in ways not explicitly considered (e.g. individuals belonging to a sibship but only considered as potential parent and offspring), and non-Mendelian transmission of genotypes (through null alleles, mutations or genotyping error).

LOD Scores

If we sample a triplet of individuals (A, B, C) with single locus genotypes gA, gB and gC, one can compare the likelihood of the hypothesis (H_1) that the three individuals are offspring, mother and father, with the likelihood of the alternative hypothesis (H_2) that the three individuals are unrelated. This comparison is usually expressed as a log-ratio, which defines the parent-pair **LOD score** (e.g. Meagher and Thompson, 1986):

$$\mathrm{LOD(gA,\ gB,\ gC)} = \log \frac{\Pr(\mathrm{gA,\ gB,\ gC} \mid H_1)}{\Pr(\mathrm{gA,\ gB,\ gC} \mid H_2)} = \log \frac{T(\mathrm{gA} \mid \mathrm{gB,\ gC})}{\Pr(\mathrm{gA})} \tag{24}$$

In this notation, the Mendelian transmission probability is denoted by T(·). The likelihood of (H_2) is the probability of observing the three genotypes when randomly drawn from a population in Hardy- Weinberg equilibrium. For diploid heterozygotes, the probability of a genotype with the alleles a_1 and a_2 and with the allele frequencies p and q is Pr(a_1, a_2) = 2pq; for homozygotes, we have Pr(a_1, a_1) = p^2.

A potential drawback of LOD scores is that if not all individuals of the population are sampled, then the total number of breeding individuals N in the population must be estimated. In order to solve this issue, Nielsen et al. (2001) proposed a Bayesian approach, extending the fractional paternity approach suggested by Devlin et al. (1988). The posterior probability that male Fi is the father of O can then be calculated for the case when the mother M is known as

$$\Pr(\mathrm{Fi} \mid \mathrm{GO,\ GM,\ GF,}\ A,\ N) = \frac{T(\mathrm{GO} \mid \mathrm{GM,\ GF}_i)}{\Sigma_j^n\, T\left(\mathrm{GO} \mid \mathrm{GM,\ GF}_j\right) + (N-n)\, T(\mathrm{GO} \mid \mathrm{GM,}\ A)} \tag{25}$$

where GO, GM, GF are the offspring, maternal and paternal genotypes, A the population allele frequencies and n the number of sampled males. So (N - n) weights the case that the true father is unsampled accordingly. Ignoring this weighting will give many false matches when the sampling rate and the amount of genomic information is low (Nielsen et al., 2001). Shamefully, in natural populations it is generally impossible to know any parent beforehand, and likelihood based methods become computationally too demanding.

Calculating the probability of a putative parent-offspring pair being false

For natural populations with few sampled parents, strict exclusion or kinship techniques are the preferred analytical approaches for parentage assignment (Jones and Ardren, 2003). Kinship methods are restrictive because they determine only whether a data set has more related individuals than expected by chance, but often cannot identify which individuals those are (Queller et al., 2000). When applied correctly, exclusion is a powerful parentage method because it fully accounts for the uniqueness of the parent-offspring relationship without any assumptions (Milligan, 2003). However, one must first determine whether their data set has enough polymorphic markers to minimize the occurrence of false pairs (i.e. adults that share an allele with a putative offspring by chance). As a consequence, many exclusion probabilities have been developed for a variety of applications.

Some approaches focus on data sets where the genotypes of the mother and putative sire, or at least one parent, are available (Chakraborty et al., 1988; Jamieson and Taylor, 1997), whereas other exclusion methods focus on excluding only a handful of candidate parents (Dodds et al., 1996). One exclusion probability that is appropriate for situations where neither parent is known was first described by Garber and Morris (1983) and later expressed in terms of homozygotes (Jamieson and Taylor, 1997).

In the present study, the probability of false parent-offspring pairs occurring within a data set is described following Christie (2009). This probability can determine whether the information content of one's data set is sufficient to accept all putative parent-offspring pairs with simple Mendelian incompatibility.

In the particular case of co-dominant markers in diploid organisms, the probability of a randomly selected pair of individuals sharing an allele from a particular locus equals

$$\Pr(Z) = \sum_{i=1}^{\mathrm{Na}} \left(2\, z_{1\,i} - z_{1\,i}^2\right)\left(2\, z_{2\,i} - z_{2\,i}^2\right) - \sum_{i=1}^{\mathrm{Na}-1} \sum_{g=i+1}^{\mathrm{Na}} \left(2\, z_{1\,i}\, q_{1\,g}\right)\left(2\, z_{2\,i}\, q_{2\,g}\right) \tag{26}$$

where Na is the total number of alleles at a locus, z_1 is the allele frequency for allele i in the sample of adults and z_2 equals the allele frequency for allele i in the sample of juveniles. Thus, z_{21} and z_{22} equal the frequency of homozygotes containing allele i in samples of adults and juveniles, respectively, assuming Hardy–Weinberg Equilibrium (HWE). Alleles occurring in only one sample (i.e. adults or juveniles) will not be included in the above expression because the product equals zero.

Notice that the expected number of homozygotes for an allele is subtracted from the total number of times the same allele occurs to prevent pairs of individuals that are homozygous for the same allele from being counted twice. Likewise, it is important to count only dyads (pairs of individuals) that are heterozygous for the same alleles only once. Therefore, we subtract a double summation where q equals the frequencies of alleles (i + 1) : Na and where $z_1 q_1$ and $z_2 q_2$ are used to calculate the HWE-expected genotype frequencies of unique heterozygotes in samples of adults and juveniles respectively.

Under some circumstances, it may be desirable to use an equation that does not employ HWE estimates of genotype frequencies. One example would be if genotype frequency estimates have high accuracy yet do not conform to HWE expectations. The equation that does not assume HWE is :

$$\Pr(Z_G) = \sum_{i=1}^{\mathrm{Na}} (2\, z_{1\,i} - \mathrm{zz}_{1\,i})\,(2\, z_{2\,i} - \mathrm{zz}_{2\,i}) - \sum_{i=1}^{\mathrm{Ng}} \left(\mathrm{zq}_{1\,i}\right)\left(\mathrm{zq}_{2\,i}\right) \tag{27}$$

where zz_1 and zz_2 equal the observed frequencies of homozygotes containing allele i in the samples of adults and juveniles, respectively, and zq_1 and zq_2 equal the observed frequencies of all unique heterozygotes, Ng, in the samples of adults and juveniles respectively. To expand this approach to multiple loci, it was assumed throughout this simulation study that loci are in linkage equilibrium and are thus independent of one another.

If the assumption of linkage equilibrium is valid, it is possible to multiply probabilities across loci such that :

$$\Pr(\delta) = \prod_{i=1}^{L} \Pr(Z)_i \tag{28}$$

where L equals the total number of loci.

To determine the approximate number of false parent-offspring pairs, Fpairs, for a given data set, Pr(δ) should be

multiplied by the total number of pairwise comparisons :

$$\text{Fpairs} = \Pr(\delta)\, n_1\, n_2 \tag{29}$$

where n_1 equals the number of adults and n_2 equals the number of juveniles. It is important to keep in mind that this is a probability, and that variance due to sampling will cause slight deviations from this quantity. However, on average, these equations predict the number of false pairs very accurately (Christie, 2009).

The importance of minimizing the number of false parent-offspring pairs depends on the study, although the utility and accuracy of any parentage analysis obviously deteriorate as the number of false parent-offspring pairs increases. If the expected number of false parent-offspring pairs is negligible (i.e. near 0), then strict exclusion can be safely used.
Here, the probability of any particular putative parent–offspring pair being false, when using strict exclusion, equals:

$$\Pr(\phi) = \frac{\text{Fpairs}}{\text{Np}} \tag{30}$$

where Np equals the observed number of putative parent-offspring pairs, which is simply calculated by summing the number of dyads that share at least one allele at all loci. Np is also equal to the total number of false parent-offspring pairs plus the total number of true parent-offspring pairs. Because $\Pr(\phi)$ equals the probability of any putative parent-offspring pair being false, one should strive to minimize this value by employing many polymorphic loci. In the present study, the probability of any putative parent-offspring pair being false $\Pr(\phi)$ will be analysed through simulations in R.

Analysis and results

Implementation of theoretical models in *Mathematica*: Hybrid zone simulations

Exact recursions describing the hybrid zone evolutionary setting have been implemented in a set of Mathematica functions (Wolfram 1996) that are available in Appendix I. The whole simulation setting is based on a previously developed package called MULTILOCUS, which was recently released by Kirkpatrick et al. (2002). These functions are appropriate for analysing selection and recombination in diploids. In particular, hybrid zones are simulated by setting up a list of populations or demes, which are allowed to exchange migrants every generation. The key components of our simulations are the number of demes, number of chromosomes per deme (so that for N=100, we actually have N/2 = 50 diploid individuals), the migration rate between demes, the number of diallelic loci to include as genotypes, the intensity of selection, and the recombination rate among loci.

Describing genotypes and populations

The genotype of an individual at position i is represented by the indicator variable X_i. With just two alleles per locus, X_i can take two values, which has been conveniently set at 0 or 1; for this special case, the frequency of allele 1 at position i is written p_i and the frequency of allele 0 as $q_i = 1 - p_i$. A fact that is useful later is that under these conventions, the expected value of X_i (averaging over all individuals in the population) is equal to p_i. When there are more than two alleles, we can choose any distinct values to distinguish the alleles.

Several functions have been developed in order to describe the genetic content of a population or set of populations based on their allele frequencies. Thus, the basic function **AlleleFrequencies** represents the different allele frequencies of the population, and allows for taking into account any deme size. **MakePopulation** gives a haploid population at linkage equilibrium, with allele frequency p; which is represented as **HaploidFrequencies** by default, even though the **NumericalModel** option can be used to give other representations (e.g. **DiploidFrequencies**). In any case, once we have characterized our population of interest, we can easily sample individuals from it by using the **MakeIndividuals** function. **MakeIndividuals** generates n random individuals from the population ψ. If ψ represents a diploid population, the representation **DiploidIndividuals** is returned (that is, we get the genotype of diploid individuals); similarly for haploids. This series of functions dealing with allele frequencies, genotype frequencies and individual genotypes will form the basis of our implementation of a cline. After all, clines can be thought to represent a series of interconnected populations that share individuals/genes through migration.

Including fitness

In our simulation scenario, selection is modelled through a fitness function, W, which gives the relative fitness of each genotype. **Absolute fitness** of a genotype is defined as the ratio between the number of individuals with that genotype after selection to those before selection. It is calculated for a single generation and may be calculated from absolute numbers or from frequencies. When the fitness is larger than 1, the genotype increases in frequency after reproduction; a ratio smaller than 1 indicates a decrease in frequency. The fitness function has the form **fitness**[deme][X,Y]. For example, f[11][{0,0,0},{1,1,1}] gives the fitness of an individual heterozygous at all 3 loci, in deme 11. The function fitness allows to specify multiplicative selection with an intensity +s to the right and -s to the left of the cline. It is worth mentioning that the efficacy of selection acting simultaneously at linked sites (a codon, a nucleotide or a gene, depending on what contributes to fitness) is reduced compared with the same selection pressure acting at independent sites (Hill and Robertson, 1966; Li, 1987). This is because linkage disequilibrium between alleles at selected loci, generated by the stochastic nature of mutation and sampling in a finite population, "interferes" with the action of selection at any one locus (Felsenstein, 1974). Therefore, it would be of interest to define which is the efect of different levels of linkage on the shape of a cline.

Linkage Disequilibrium and Recombination

In population genetics, linkage disequilibrium (LD) is the non-random association of alleles at two or more loci. In other words, linkage disequilibrium describes a situation in which some combinations of alleles or genetic markers occur more or less frequently in a population than would be expected from a random formation of haplotypes from alleles based on their frequencies. Non-random associations between polymorphisms at different loci are measured by the degree of linkage disequilibrium (D). D indicates the deviation of the observed frequency of a haplotype from that expected if the alleles at two loci were independent from each other, so that two loci, A and B, are said to be in linkage (or gametic) disequilibrium if their respective alleles do not associate independently. In the present study, and following the MULTILOCUS implementation, a matrix of pairwise linkage disequilibrium for the population (ψ) is obtained through the **DisequilibriumTable** function. It also applies to a list of demes or populations, representing a cline. It returns the D between selected loci, and the D between the neutral and selected loci. The **DisequilibriumMean** function gives the mean pairwise linkage disequilibrium for the population. Again, it also applies to a list of populations, representing a cline.

Linkage disequilibrium arises as a consequence of three features of life a) the physical structure of chromosomes; b) the inherent mutations that occur at random during DNA replication; c) the rate of recombination between any two given loci. Taking each in turn, this means that markers or genes do not undergo independent assortment if they are on the same chromosome. This means that when a new mutation arises it will be inherited along with all of the other markers/polymorphisms that occur on that chromosome. Unless of course a recombination event occurs between two loci that serves to break the pattern of mutations that are inherited on one chromosome. Genetic recombination is a process by which a molecule of nucleic acid (usually DNA) is broken and then joined to a different DNA molecule. It is equivalent to "allele-shuffling", since it makes new allele combinations to appear. In the present study, the amount of recombination between markers is defined by the option **Linkage**, which specifies a linear genetic map.

Simulating a cline

As previously stated, a Cline can be represented by a series of interconnected populations that share individuals/genes through migration. Individuals at opposite extremes of a linear Cline will be under different selection regimes. Those individual alleles which are favoured in one part of the cline, will be selected against on the other side. Thus, the frequency of different alleles will change gradually while moving along the Cline. In our *Mathematica* implementation, the **MakeCline** function sets up a stepped cline with any number of haploid demes, genes, or individuals in each deme. The user can define a linear gradient for the cline, spanning Δp, and can easily obtain the cline widths at each locus through the **ClineWidth** function. Thanks to the **IterateExact** and **StoreExact** simulation functions, the user is allowed to trace the evolution of any number of demes influenced by several selection regimes, different migration rates and the presence/absence of linkage between markers.

Simulating a cline: Results

Setting up a stepped cline across 20 demes, with 3 loci in each:

Each deme is represented by a list of genotype frequencies. The deme size is arbitrary and does not affect the results.

Fitness is given by a function of the form f[deme][X,Y]. For example, f[11][{0,0,0},{1,1,1}] gives the fitness of an individual heterozygous at all 3 loci, in deme 11.
The function fitness[s,mid,ecotone] specifies multiplicative selection +s to the right, -s to the left

```
fitness[s_, mid_, ecotone][i_][X_, Y_] :=
  If[i > mid, (1 + s)^(Plus@@(X+Y)-Length[X]), (1 + s)^-(Plus@@(X+Y)-Length[X])];
```

For example, this tabulates the fitness of genotype {{0,0,0},{0,0,0}} across the cline, for s = 0.1 and with midpoint at deme 10:

```
Table[fitness[0.1, 10, ecotone][i][{0, 0, 0}, {0, 0, 0}], {i, 20}]
```

```
{1.331, 1.331, 1.331, 1.331, 1.331, 1.331, 1.331, 1.331, 1.331, 1.331, 0.751315, 0.751315,
  0.751315, 0.751315, 0.751315, 0.751315, 0.751315, 0.751315, 0.751315, 0.751315}
```

We can easily allow for different selection intensities on different loci by letting s be a list $\{s_1, s_2, s_3\}$:

```
fitness[s_List, mid_, ecotone][i_][X_, Y_] :=
  If[i > mid, Times @@ (1 + s)^(Plus@@(X+Y)-Length[X]), Times @@ (1 + s)^-(Plus@@(X+Y)-Length[X])];
```

The fitness changes from $(1 + s)^3$ to $(1 - s)^3$

Note: In the simulations, fitnesses of each diploid genotype were calculated once, and then stored. This made calculations much faster.

Simulating a cline : Cline shape

This simulates the cline with stepped selection s = 0.01 on the three unlinked loci, across 30 demes. Migration rates between adjacent demes is $m = 0.5$ and results are stored at t = 0, 20, …500 generations.

```
fitness[s_, mid_, ecotone][i_][X_, Y_] :=
  If[i > mid, Times @@ (1 + s)^(Plus@@(X+Y)-Length[X]), Times @@ (1 + s)^-(Plus@@(X+Y)-Length[X])];
StoreExact[res, MakeCline[30, 3, 100], fitness[0.01, 15, ecotone], 0.5, {500, 20},
  Compiled → True, FixedEnds → {MakePopulation[3, 0, 100], MakePopulation[3, 1, 100]}];
```

Compiled→True compiles the code, which is much faster, while FixedEnds→{pop0,pop1} fixes allele frequencies at the ends. Note that the end demes have to be the same size as the demes in the main population.

We can now describe the state of the cline at t = 500. That is, after 500 generations. For example, we can obtain the allele frequencies for each deme, which, by symmetry, are the same at all three loci.

```
AlleleFrequencyTable[res[500]] // TableForm
```

0.0102594	0.0102594	0.0102594
0.020942	0.020942	0.020942
0.032467	0.032467	0.032467
0.0452755	0.0452755	0.0452755
0.0598474	0.0598474	0.0598474
0.0767147	0.0767147	0.0767147
0.0964756	0.0964756	0.0964756
0.119806	0.119806	0.119806
0.147469	0.147469	0.147469
0.180322	0.180322	0.180322
0.219313	0.219313	0.219313
0.265468	0.265468	0.265468
0.319862	0.319862	0.319862
0.383563	0.383563	0.383563
0.457533	0.457533	0.457533
0.542467	0.542467	0.542467
0.616437	0.616437	0.616437
0.680138	0.680138	0.680138
0.734532	0.734532	0.734532
0.780687	0.780687	0.780687
0.819678	0.819678	0.819678
0.852531	0.852531	0.852531
0.880194	0.880194	0.880194
0.903524	0.903524	0.903524
0.923285	0.923285	0.923285
0.940153	0.940153	0.940153
0.954724	0.954724	0.954724
0.967533	0.967533	0.967533
0.979058	0.979058	0.979058
0.989741	0.989741	0.989741

This gives the average:

```
AlleleFrequencyMean[res[500]]
```

```
{0.0102594, 0.020942, 0.032467, 0.0452755, 0.0598474, 0.0767147,
  0.0964756, 0.119806, 0.147469, 0.180322, 0.219313, 0.265468, 0.319862, 0.383563,
  0.457533, 0.542467, 0.616437, 0.680138, 0.734532, 0.780687, 0.819678, 0.852531,
  0.880194, 0.903524, 0.923285, 0.940153, 0.954724, 0.967533, 0.979058, 0.989741}
```

This shows how the cline changes over time, getting more sigmoideal, thanks to dispersal/gene flow (Fig. 4).

```
AllCline = Table[ListPlot[AlleleFrequencyMean[res[t]], Joined → True, PlotRange → {{0, 30}, {0, 1}},
      PlotStyle → RGBColor[t / 150, t / 850, t / 600, 0.9]], {t, 0, 500, 100}];
Show[AllCline]
```

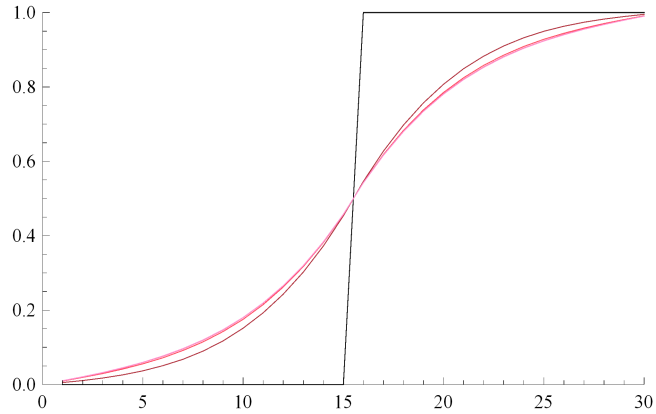

Figure 4. Evolution of the cline shape through time. From t = 0 (black) to t = 500 (purple).

Cline width reaches equilibrium at w = 11.7 for all 3 loci within less than 400 generations

Table[{t, ClineWidth[res[t]]} // Flatten, {t, 0, 500, 20}] // TableForm

0	1	1	1
20	6.77627	6.77627	6.77627
40	8.65734	8.65734	8.65734
60	9.70305	9.70305	9.70305
80	10.354	10.354	10.354
100	10.7833	10.7833	10.7833
120	11.0765	11.0765	11.0765
140	11.2808	11.2808	11.2808
160	11.4247	11.4247	11.4247
180	11.5266	11.5266	11.5266
200	11.5988	11.5988	11.5988
220	11.6501	11.6501	11.6501
240	11.6865	11.6865	11.6865
260	11.7123	11.7123	11.7123
280	11.7306	11.7306	11.7306
300	11.7436	11.7436	11.7436
320	11.7527	11.7527	11.7527
340	11.7592	11.7592	11.7592
360	11.7638	11.7638	11.7638
380	11.767	11.767	11.767
400	11.7693	11.7693	11.7693
420	11.771	11.771	11.771
440	11.7721	11.7721	11.7721
460	11.7729	11.7729	11.7729
480	11.7735	11.7735	11.7735
500	11.7739	11.7739	11.7739

Variations in cline shape can be seen most easily by plotting allele frequency ratios on a *logit* scale, $\text{Log}\left[\frac{p}{q}\right] = \text{Log}\left[1 \Big/ \left(\frac{1}{p} - 1\right)\right]$. The logit function is the inverse of the "sigmoid", or "logistic" function commonly used in mathematics (showed in Fig. 1). With the following command, we plot the cline maintained by stepped selection across an "ecotone" generated in the last section:

```
ListLogPlot[1 / (1/AlleleFrequencyMean[res[500]] - 1), Joined → True, Axes → True]
```

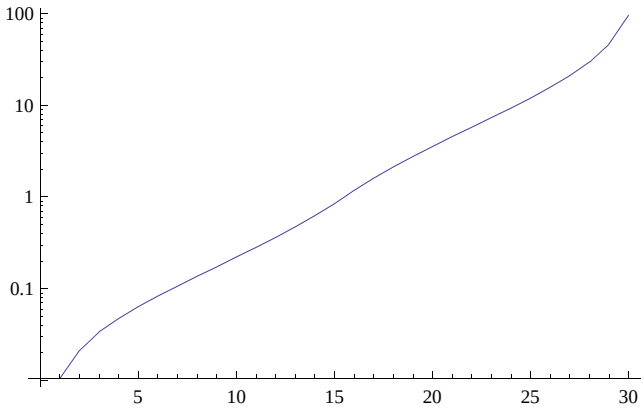

Figure 5. LogPlot showing the cline shape with selection intensity s = 0.01 and migration mig = 0.5.

Linkage Disequilibrium and Recombination

Again, linkage disequilibrium describes a situation in which some combinations of alleles or genetic markers occur more or less frequently in a population than would be expected from a random formation of haplotypes from alleles based on their frequencies. This shows the pattern of linkage disequilibrium between three unlinked loci in our simulated cline:

```
ListPlot[DisequilibriumMean[res[500]], Joined → True]
```

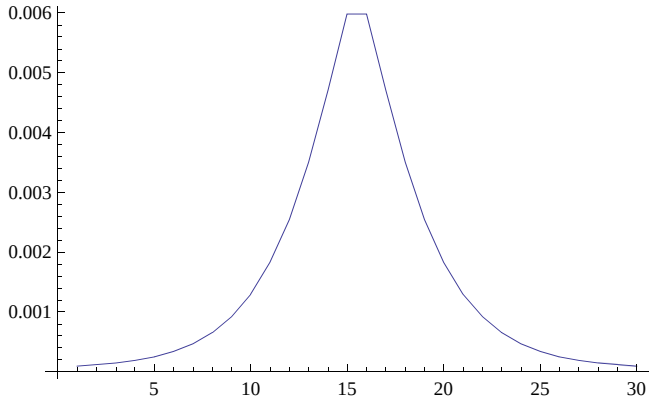

Figure 6. Distribution of linkage disequilibrium between three unlinked loci along a simulated cline with selection intensity s = 0.01 and migration mig = 0.5.

It is at the center of the cline, where mixing of individuals from different species is more intense, that the strength of the disequilibrium is higher. In fact, the strength of disequilibrium in the centre is ~25% of the maximum possible disequilibrium, pq:

```
centercline = DisequilibriumMean[res[500]][[15]]
```

0.00598569

```
pp = AlleleFrequencyMean[res[500]][[15]];
maxdis = DisequilibriumMean[res[500]][[15]] / (pp (1 − pp))
```

0.0241168

```
centercline / maxdis
```

0.248197

When loci are in the same chromosome, we say that they are linked. Only recombination can break down the joint segregation of linked markers. In order to analyze the impact of recombination on the cline, we can set up simulations for different recombination levels r = {0.05, 0.1, 0.2, 0.5}:

```
(StoreExact[res[3, 0.04, #], MakeCline[30, 3, 100], fitness[0.04, 15, ecotone], 0.5, {500, 20},
    Linkage → {#, #}, Compiled → True,
    FixedEnds → {MakePopulation[3, 0], MakePopulation[3, 1]}];) & /@ {0.05, 0.1, 0.2, 0.5};
```

And now we show how the pattern of LD is reduced for increasing levels of recombination through a combined plot:

```
AllCline = Table[ListPlot[DisequilibriumMean[res[3, 0.04, t][500]], Joined → True,
    PlotRange → {{0, 30}, {0, 0.12}}, PlotStyle → RGBColor[12 t^2, 2 t^2, 20 t^2, 0.9]],
  {t, {0.05, 0.1, 0.2, 0.5}}];
Show[AllCline]
```

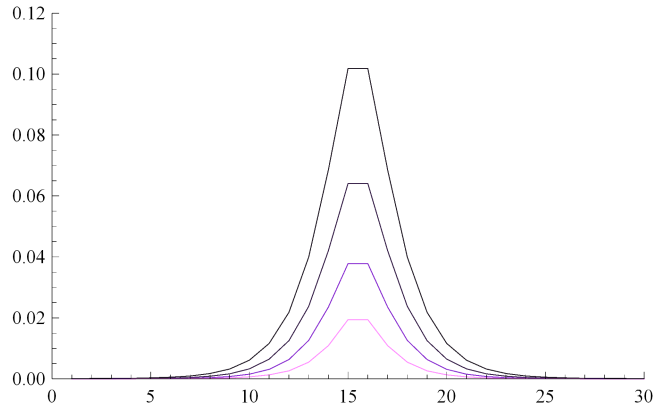

Figure 7. Change on levels of linkage disequilibrium with different recombination rates: r = {0.05, 0.1, 0.2, 0.5}.

The value of D at the centre of the cline (deme number 15) for different levels of recombination will be:

```
DisequilibriumMean[res[3, 0.04, #][500]][[15]] & /@ {0.05, 0.1, 0.2, 0.5}
```

{0.101872, 0.0641421, 0.0378334, 0.0194127}

Simulating clines with different selection coefficients : Cline Width

In this case, results for a cline with selection intensity s = 0.08 are stored in res[3, 0.8][t]

```
StoreExact[res[3, 0.08], MakeCline[30, 3, 100], fitness[0.01, 15, ecotone], 0.5, {500, 20},
  Linkage → {0.1, 0.1}, Compiled → True,
  FixedEnds → {MakePopulation[3, 0, 100], MakePopulation[3, 1, 100]}];
```

```
ListPlot[DisequilibriumMean[res[3, 0.08][500]], Joined → True]
```

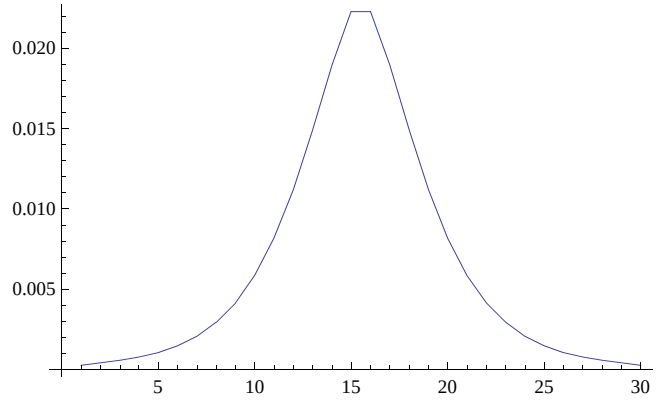

Figure 8. Distribution of linkage disequilibrium between three loci along a simulated cline with selection intensity s = 0.08.

It can be observed that the levels of Linkage Disequilibrium along the cline are more than 3 times higher than previously obtained (Fig. 6)-

We finally simulate the cline under different selection levels, ranging from 0.01 to 0.12:

```
fitness[s_, mid_, ecotone][i_][X_, Y_] :=
    If[i > mid, Times @@ (1 + s)^Plus@@(X+Y)-Length[X], Times @@ (1 + s)^-(Plus@@(X+Y)-Length[X])];
(StoreExact[res[3, #, ecotone], MakeCline[30, 3, 100], fitness[#, 15, ecotone], 0.5, {500, 20},
          Compiled → True, FixedEnds → {MakePopulation[3, 0], MakePopulation[3, 1]}];) & /@
    {0.01, 0.02, 0.04, 0.08, 0.12};
```

It can be observed that with increasing selection pressure the cline gets steeper, so that the transition between one allele type to the other is sharper.

```
AllCline = Table[ListPlot[AlleleFrequencyMean[res[3, t, ecotone][500]], Joined → True,
        PlotRange → {{0, 30}, {0, 1}}, PlotStyle → RGBColor[12 t^2, 2 t^2, 20 t^2, 0.9]],
    {t, {0.01, 0.02, 0.04, 0.08, 0.12}}];
Show[AllCline]
```

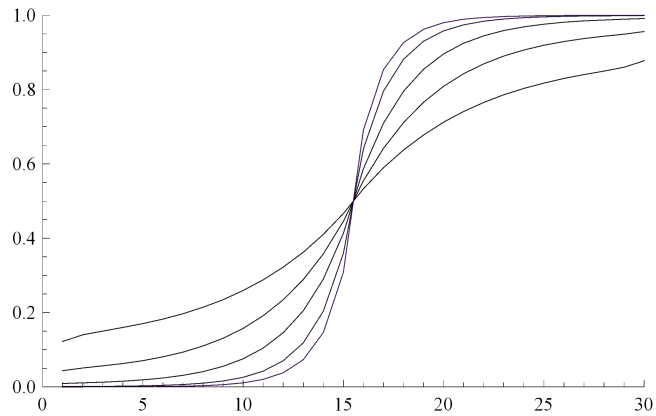

Figure 9. Change on cline width at different selection levels.

```
Flatten[{{#, ClineWidth[res[3, #, ecotone][500]], Sqrt[6 × 0.5 / (2 #)]}} & /@ {0.01, 0.02, 0.04, 0.08, 0.12} //
 TableForm
```

0.01	15.0678	15.0678	15.0678	12.2474
0.02	9.15416	9.15416	9.15416	8.66025
0.04	5.75431	5.75431	5.75431	6.12372
0.08	3.55105	3.55105	3.55105	4.33013
0.12	2.61478	2.61478	2.61478	3.53553

The cline widths for the three loci decrease as selection gets stronger, but are still narrower than predicted (5th column) for larger selection levels (4-5th row). This may be because the diffusion approximation is accurate when selection becomes very weak, while in here selection gets fairly strong.

Pedigree simulations in *Mathematica* and Parentage analysis in R.

Creating a pedigree

When building the pedigrees, we assume that each deme along the cline corresponds to a population of N diploid individuals. The pedigree spanning t generations is represented by a sequence of $N \times N$ matrices, M_0, M_1, …, M_t; throughout, we count time back into the past. For the present study we will focus on reconstructing parentage, for which only one generation will be traced. In any case, the i'th row of M_t specifies the parents of individual i in generation t, so that the matrix M_t connects generation t with $t+1$ (counting backwards in time). If an individual has two different parents, then the row has two non-zero elements, set at 1; if it is produced by self-fertilisation, then there is a single non-zero entry, with value 2 (Fig. 10). The matrix is represented in *Mathematica* as a sparse array (Wolfram, 1996), which allows large populations ($N \sim 1000$) to be handled efficiently, since only $2N$ elements are stored, rather than N^2.

-

$$\begin{pmatrix} 1&0&0&1&0&0&0&0&0&0&0&0\\ 1&0&1&0&0&0&0&0&0&0&0&0\\ 0&0&1&0&0&0&0&0&0&0&1&0\\ 0&1&1&0&0&0&0&0&0&0&0&0\\ 0&0&0&0&0&1&0&0&0&0&0&1\\ 0&0&1&0&0&1&0&0&0&0&0&0\\ 0&0&0&0&0&1&1&0&0&0&0&0\\ 0&0&0&0&2&0&0&0&0&0&0&0\\ 0&0&0&0&0&1&0&0&0&0&0&1\\ 0&0&0&0&0&0&0&0&1&0&1&0\\ 0&0&0&0&0&0&0&0&1&1&0&0\\ 0&0&1&0&0&0&0&0&0&0&0&1 \end{pmatrix} \quad \begin{pmatrix} 0&0&1&0&0&0&1&0&0&0&0&0\\ 0&1&0&1&0&0&0&0&0&0&0&0\\ 0&1&1&0&0&0&0&0&0&0&0&0\\ 0&2&0&0&0&0&0&0&0&0&0&0\\ 0&0&0&0&0&2&0&0&0&0&0&0\\ 0&0&0&0&0&2&0&0&0&0&0&0\\ 0&0&1&0&1&0&0&0&0&0&0&0\\ 0&1&0&0&0&1&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0&0&0&1&1\\ 0&0&0&0&0&0&0&0&0&2&0&0\\ 0&0&0&0&0&0&0&0&1&0&0&1\\ 0&0&0&0&0&0&0&0&1&1&0&0 \end{pmatrix}$$

Figure 10. Pedigree matrices for 3 demes with 8 individuals each under the Island Model (left) or the Stepping Stone Model (right) of pollen dispersal.

In the most general neutral model used in population geneteics, the Wright-Fisher model, each parent is chosen at random, with replacement, so that a fraction averaging $1/N$ are produced by selfing. Parents are chosen independently, with probability proportional to their individual fitness. Once the pedigree is determined, genotypes are then chosen randomly. With discrete unlinked loci, each diploid parent passes on one or other of its genes with equal probability, independently across loci. Genes may be labelled 0 or 1 to indicate their allelic state, or they may be given a unique integer in the first generation, so that identity by descent can be followed ('gene dropping'; Edwards, 1968; McCluer et al., 1986).

In the present study, several new functions have been developed in order to deal with reproduction and pedigree construction in a simulated cline. First of all, the function **SparsePedigreeMatrix** gives a random matrix showing the mating pattern between all the individuals within a deme. This function also allows for selection according to the infinitesimal model. The infinitesimal model assumes a very large (effectively infinite) number of loci each with infinitesimal effect. Of course, the infinitesimal model can not be taken as an exact description of biological reality, but

when a large number of loci underlie a character, the infinitesimal model provides a very satisfactory treatment of short term response. Under the infinitesimal model, the amount of selection acting on any given locus is expected to be very small, and hence the expected change in allele frequencies over a few generations is also very small. Once we have the genotypes from our parental population (individuals sampled from each deme) and the corresponding pedigree matrix, we can generate genotypes for the offspring using the **DropGenes** function. **DropGenes** uses the **Mendel** function to get a random gamete, assuming no linkage. Finally, **Parents** returns the indices for the parents of each individual in a pedigree.

Estimating Exclusion Probability

The exclusion probability method is included in the present report through an R script that calculates the probability of any putative parent-offspring pair being false (Appendix II). The R script provided is a modified version of a set of scripts recently developed by Christie (2009). Before running this script you need to make sure that you set the working directory within the script to a valid location so that R can read your input files and create output files (e.g., setwd("C:/EXCLUSION")). This is easily accomplished by creating a folder in your C: drive called "EXCLUSION"-the default directory used throughout the script. Furthermore, the input files should be placed within this folder.

The input files have a simple, but obligate format. They correspond to the two tab-delimited text files created by our *Mathematica* simulations, one for adults specifically named "adults" and the other for juveniles specifically named "juveniles". The input files should be saved as text files within the specified directory folder (e.g. "C:/EXCLUSION") and have the first row with the following headers in each column: ID, Locus1a, Locus1b, Locus2a, Locus2b etc. It is important that the diploid data from each locus are side by side starting at column 2.

Once input files are in the correct format, we can calculate the probability of any putative parent-offspring pair being false by running our R script. After running this script you will obtain three main types of output files called:

1) "PrZ" provides you with the per locus exclusion probabilities (equation 26).
2) "Fpairs+Prdelta" provides you with the expected number of false pairs in your data set and Pr(δ) (equations 29 and 28, respectively).
3) "Phi" provides you with Pr(Φ) (equation 30)-the probability of any putative parent-offspring pair being false.

Finally, the Putative pairs can be used to compare the R script results with the Pedigree structure simulated using *Mathematica*. Please, note that in the R code presented below (Appendix II), both text files are created independently of *Mathematica*, in order to simplify the presentation.

Pedigree simulations in *Mathematica* and Parentage analysis in R: Results

Pedigree simulations in *Mathematica*

This simulates a three loci cline with 40 demes and creates genotypes for 50 individuals each. Migration rates between adjacent demes is m = 0.5. Results are stored at t = 0, 20, …500.

The key parameters are:

```
ndemes = 40; (*Number of demes*)
nind = 100;(*Number of chromosomes per deme. 100 = 50 diploid individuals*)
nloci = 3; (*Number of diallelic loci to include*)
recomb = 0.1;(*Recombination rate*)
linkmap = Table[recomb, {nloci - 1}];(*We create a list to pass to the Linkage option*)
mig = 0.5 (*Migration rate between demes*)
```

This defines a fitness function and evaluates it for the homozygote [{0, 0, 0}, {0, 0, 0}]. Then it simulates the cline with stepped selection s = 0.03 on the three loci (nloci defined above), across 40 demes (ndemes defined above).

```
fitness[s_, mid_, ecotone][i_][X_, Y_] :=
    If[i > mid, (1 + s)^(Plus@@(X+Y)-Length[X]), (1 + s)^-(Plus@@(X+Y)-Length[X])];
Table[fitness[0.1, 20, ecotone][i][{0, 0, 0}, {0, 0, 0}], {i, ndemes}];
```

```
StoreExact[new, MakeCline[ndemes, nloci, nind], fitness[0.03, 20, ecotone], mig, {50, 2},
    Compiled → True,
    FixedEnds → {MakePopulation[nloci, 0, ndemes], MakePopulation[nloci, 1, ndemes]}];
ψ = new[50];
```

With this function we create a list containing the genotypes of 50 individuals from the whole cline sampled after 500 generations. This will form our parental population.

```
popnews = MakeIndividuals[ψ, Table[nind, {ndemes}]];
```

Now we can define the pedigree matrix for any specific deme within the cline. For example, this creates a PedigreeMatrix for 50 individuals and genotypes for 50 descendants from deme number 20.

```
pop1 = popnews[[20]][[1]];
ped1 = SparsePedigreeMatrix[nind];
off1 = DropGenes[pop1, ped1];
pop1 = HaploidIndividuals[pop1];
off1 = HaploidIndividuals[off1];
```

This does the same but for the whole cline simultaneously. Note that we are calling Mendel directly without going through DropGenes:

```
pop1 = Table[popnews[[i]][[1]], {i, ndemes}];
ped1 = Table[SparsePedigreeMatrix[nind], {ndemes}];

OffsCline = Table[Mendel[pop1[[i]][[#]]] & /@ Parents[ped1[[i]]], {i, ndemes}];
Dimensions[OffsCline]
```

```
{40, 100, 3}
```

Effectively, we get alleles from 3 different markers in 40 demes with 100 haploid individuals (chromosomes) each.

```
pop1[[1]]; OffsCline[[1]]; pop1[[21]]; OffsCline[[21]]; pop1[[40]]; OffsCline[[40]];
pop1 = Table[HaploidIndividuals[pop1[[i]]], {i, ndemes}];
off1 = Table[HaploidIndividuals[OffsCline[[i]]], {i, ndemes}];
MakeDiploid[pop1][[23]];
MakeDiploid[off1][[23]];
```

Finally, we just need to Export our genotype table to a convenient text file, which will be used as input for the Exclusion Probability R scripts.

```
MakeDiploid[pop1][[23]][[1]];
Length[Flatten[MakeDiploid[pop1][[23]][[1]]]];
TableForm[Partition[Flatten[MakeDiploid[pop1][[23]][[1]]], nloci]];
Indlabels =
    Flatten[Transpose[{Table["Ind−" <> ToString[i], {i, nind / 2}],
          Table["Ind−" <> ToString[i], {i, nind / 2}]}]];
Pargenotypes =
    Table[Join[Flatten[{"Individual", Table["Locus−" <> ToString[i], {i, nloci}]}],
       Flatten[Transpose[{Indlabels, Partition[Flatten[MakeDiploid[pop1][[j]][[1]]], nloci]}]]],
      {j, ndemes}];
Filename = Table["Parents−" <> ToString[i] <> ".txt", {i, ndemes}];
Do[Export[ToString[Filename[[n]]], TableForm[Partition[Pargenotypes[[n]], nloci + 1]],
      "Table"], {n, ndemes}];
```

And we do the same for the Offspring genotypes.

```
MakeDiploid[off1][[23]][[1]];
Length[Flatten[MakeDiploid[off1][[23]][[1]]]];
TableForm[Partition[Flatten[MakeDiploid[off1][[23]][[1]]], nloci]];
Indlabels =
    Flatten[Transpose[{Table["Ind−" <> ToString[i], {i, nind / 2}],
          Table["Ind−" <> ToString[i], {i, nind / 2}]}]];
Offgenotypes =
    Table[Join[Flatten[{"Individual", Table["Locus−" <> ToString[i], {i, nloci}]}],
       Flatten[Transpose[{Indlabels, Partition[Flatten[MakeDiploid[off1][[j]][[1]]], nloci]}]]],
      {j, ndemes}];
Filename = Table["Offspring−" <> ToString[i] <> ".txt", {i, ndemes}];
Do[Export[ToString[Filename[[n]]], TableForm[Partition[Offgenotypes[[n]], nloci + 1]], "Table"],
    {n, ndemes}];
```

Thus, we have a series of genotypes following the allele freuquencies imposed by the specific selection-gene flow pattern. Remember, we set up a cline with stepped selection s = 0.03 on three loci and with migration rates between adjacent demes mig = 0.5.

Parentage analysis in R

Despite being able to simulate parent-offspring relationships within a simulated hybrid zone, the analysis of parental exclusion was carried out in a simplified setting, given the limitations in time and computational resources. The simplified setting, which could be used as a minimum base-line for the efficiency of molecular markers, consists of a single deme with allele frequencies following the Hardy-Weinberg equilibrium conditions.

The number of loci used for this simulation study ranged from 10 to 20 molecular markers, which is within the standard number used in current paternity studies. Each molecular marker was allowed to be assigned a specific allele frequency distribution, namely the Bernatchez, the Geometric or the Uniform distribution (Appendix II).

In the present study, neutral alleles were sampled from a geometric distribution, since this type of distribution is commonly found in nature. The geometric distribution is the distribution of the total number of trials before the first success occurs, where the probability of success in each trial is *p* (Fig. 11).

```
ListPlot[Table[PDF[GeometricDistribution[0.3], k], {k, 0, 30}], PlotRange → {{0, 30}, {0, 0.30}},
  Filling −> Axis, Axes → True]
```

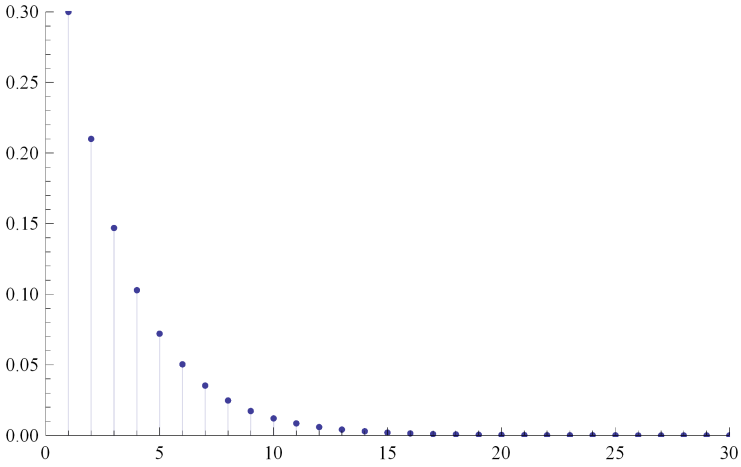

Figure 11. A geometric distribution with probability parameter p = 0.3.

This distribution of allele frequencies corresponds to the fact that only a few alleles are present in a significant proportion of individuals, while most alleles are only present in a few individuals.

Our simulations allowed to define the index of individuals that produce offspring, which will be the ones that contribute to the next generation. These indices correspond to the pedigree matrices previously described. Finally, a proportion of the offspring population (False Offspring) is sampled from a new population, with a different allele frequency distribution.

The calculation of Pr(Z) and Pr(δ) as defined in equations (26) and (28) was carried out as indicated in the Methods section. Only the alleles that are found both in the parentals and in the offspring were considered. It is important to note that alleles occurring in only one sample (i.e. adults or juveniles) do not need to be included in the calculation because their product equals zero.

Estimates of the probability of any putative parent–offspring pair being false (Phi; equation 30) were obtained through the R script included in Appendix II. Moreover, 10 replicates per parameter combination were carried out, in order to get a rough estimate of the variability found between different runs.

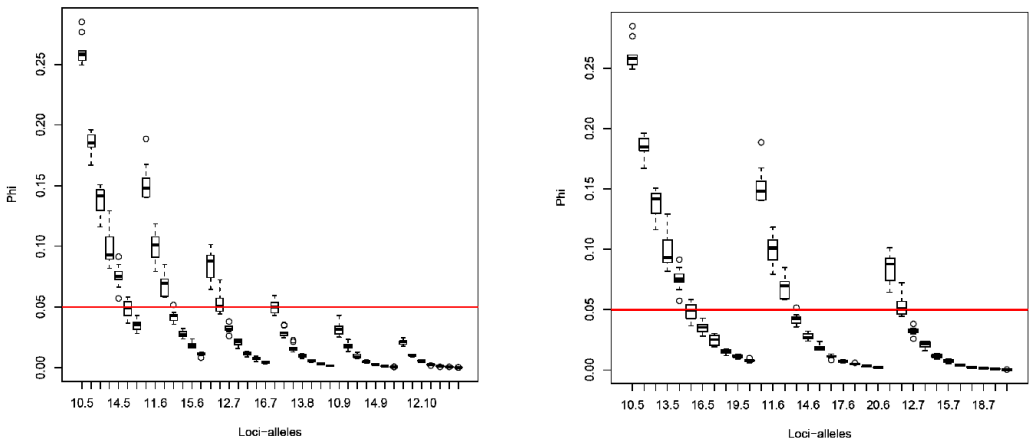

Figure 12. Estimated probability of any putative parent–offspring pair being false using a) 5 to 10 alleles in 10 to 16 loci or b) 5 to 7 alleles in 10 to 20 loci.

The red line at 0.05 in figure 12 places the limit at which we would have 0.95 confidence in our parent-offspring assignments (= 0.05 False pairs). That is, it would mean we are accepting False parent-offspring pairs with low probability (which is what we want when doing parentage analysis). In the figure 12a, you can see 6 groups (5 to 10 alleles) and 7 boxplots per group (10 to 16 loci). It is observed that if all markers had 5 alleles only (first group of replicates), we would need at least 15 markers to get 0.95 confidence in our parent-offspring assignments (= 0.05 False

pairs). In the figure 12b, we can see 3 groups (5 to 7 alleles) and 11 boxplots per group (10 to 20 loci). In this plot it is highlighted that the rate of decrease in Phi slows down when we get to ~18 markers with 5 alleles each or ~16 markers with 6 alleles each.

Discussion

Hybrid zone simulations

Hybrid zone simulations: Discussion of results

Our results indicate that, for a given level of migration between demes, different levels of selection have a direct impact on the cline width, with stronger natural selection causing clines to be narrower. The analysis of the temporal changes in cline shape shows how the cline gets a typical sigmoideal form over time until reaching a gene flow-selection equilibrium (Fig. 2). Our results also showed that the intensity and pattern of natural selection along the cline influences not only the shape of the cline, but also the velocity at which the stable equilibrium is reached. Even with extremely high migration rates such as those included in the present study (m = 0.5), equilibrium is reached in about 400 generations. It should be considered that neutral alleles will take ~10,000 generations to spread over ~100σ by diffusion alone (see below), for which patterns of selected loci will generally be different to those observed in neutral markers.

Interestingly, our simulations pointed out the fact that selection also plays a significant role on the distribution of the level of linkage disequilibrium along the cline. Much work has been done on the interaction of gene flow with spatially varying selection on a single locus (see Slatkin, 1973), but not with multiple loci. There is an interesting question which arises in this problem: How much linkage disequilibrium between loci can be produced by gene flow? The presence of linkage disequilibrium has been thought to be a sensitive measure of additive epistasis between loci (Lewontin, 1974). However, our results show that a large amount of linkage disequilibrium can be generated by gene flow in a cline, even in the absence of epistasis. This same point has been made by Li and Nei (1974) and by Prout (1973) for other models of gene flow.

Hybridization introduces sets of alleles, which gradually disperse through sucessive backcrosses, by segregation and recombination. Alleles which entered either population many generations back should by now have reached linkage equilibrium. In principle, we could estimate the rate of hybridization over the past few generations from the linkage disequilibrium, or in other words, by estimating the excess of individuals carrying multiple introgressed alleles. One approach would be to classify each individual as being a first, second, or later generation backcross, according to whether they carry 1/4, 1/8, … of their genome introgressed (Nason and Ellstrand, 1993; Boecklen and Howard, 1997). However, this would only be accurate with an extremely large number of loci. Instead, a plausible way forward around this problem would be to first establish if there is a significant excess of 'complex' hybrids, and then use maximum likelihood to estimate the rate of introgression and the degree of ancestral polymorphism. Such an interesting approach remains as an open question for further studies.

Hybrid zone simulations: Limitations of the model

Despite the apparent complexity of the diffusion approximation models in population genetics, they still show several limitations, since represent a simplified version of reality. The main limitations of the model of hybrid zones presented here are due to the unrealistic assumptions upon which the diffusion approximation is based, namely the uniform distribution of individuals and random migration following a symmetrical distribution. Since the natural environment is heterogeneous per se, individuals tend to be present in a non-uniform distribution, and this should be taken into account in the models. Moreover, main wind direction, pollinators' behaviour, water rainoff of seeds...all these would make migration not to be symmetrical (unless all these forces act in oposite directions and cancel out), for which a different approach is needed.

It should be noticed that the use of the analogy of physical diffusion will only be satisfactory when the distances of dispersion in a single generation are small compared with the length of the wave. In reality diffusion is a complex process, compounded of the diffusion of gametes, larvae, and adults; a more exact treatment than that supplied by a simple coefficient would involve the interaction of these components, and the stages at which the selective advantage was enjoyed. So far as it is applicable, the analogy of physical diffusion, therefore, greatly simplifies the problem. With respect to the assumed independence of selection from allele frequency, this is effectively to assume that there is no

dominance in respect of the selective advantage enjoyed. Apart from its simplicity this is also the most important case to consider, in respect to advantageous mutations occurring in nature. There are, at least, plausible reasons for supposing that the common recessiveness of observed mutations is a characteristic of harmful mutations, which have long been appearing in the species with relatively high mutation rates, whereas beneficial mutations must, at the time of their establishment, occur with exceedingly low mutation rates, and have rarely appeared before in the recent history of the species. On these grounds, dominance would be expected to be absent, and its absence is made more likely by the fact that in most cases the quantitative effect of beneficial mutations must be extremely small. For the same reason the selective intensity is taken to be a small quantity, so that the allele frequency (p) may be taken to vary continuously with time, from generation to generation.

Although gene frequency can change abruptly across the centre of a hybrid zone, foreign alleles can still penetrate far into either side. It is this gene flow between distinct populations that binds them together into one biological species. Smooth sigmoid clines are expected for introgressing alleles that are diffusing freely or are subject to only weak selection. A barrier to gene flow, caused for example by a physical obstacle, will produce a step in gene frequency, Δp, at the centre of the cline proportional to the gradient (dp/dx) on either side. The strength of this barrier to gene flow can be estimated from the shape of the cline as $B = \Delta p/(dp/dx)$.

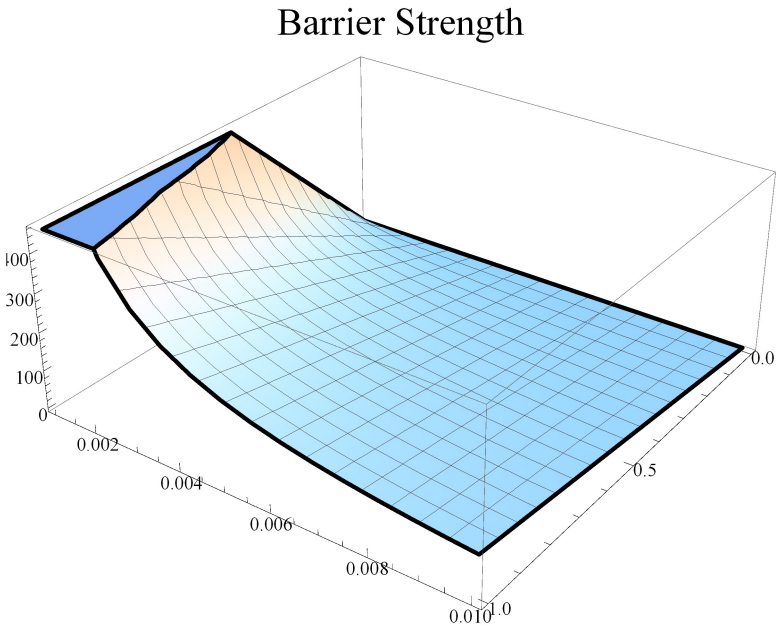

Figure 13. Change in barrier strength depending on step $(0 < \Delta p < 1)$ and gradient $\left(0.001 < \frac{\delta p}{\delta x} < 0.01\right)$ in gene frequency.

The barrier strength measure has the dimensions of distance (it can be considered as the equivalent unoccupied habitat that would have a similar effect on gene flow reduction), and its magnitude relative to the dispersal rate (B/σ) determines the delay to the spread of genes across the barrier (Fig. 14). Although neutral alleles can suffer a lengthy delay [$T \approx (B/\sigma)^2$], alleles with even a slight advantage (s) will hardly be impeded ($T \approx \log[(B/\sigma)^2 \pi s/2] / 2s$). This is because once a few of them penetrate the barrier, they will increase exponentially and spread to fixation. In making these comparisons it is important to note that even without barriers, neutral alleles will take ~10000 generations to spread over ~100σ by diffusion alone (Barton, 1979).

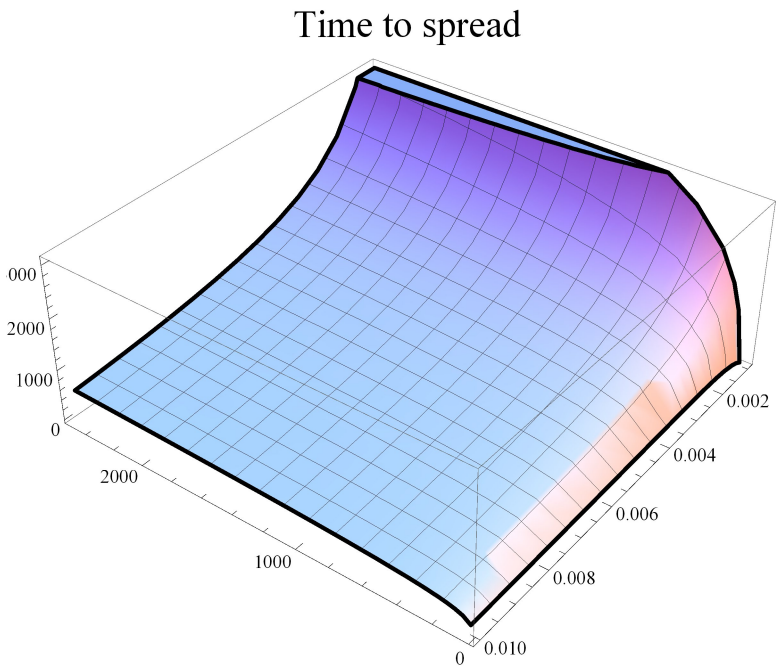

Figure 14. Delay to the spread of genes across the barrier depending on strength of the barrier $\left(\frac{B}{\sigma} = 5 - 50\right)$ and selection levels ($s = 0.001 - 0.01$).

Because species' ranges are unlikely to remain stable for such long periods, neutral variants are much more likely to be carried long distances by gross population movements ('convection') than by local diffusion. Therefore, the ability of the diffusion approximation to explain long-term evolutionary changes is highly compromised.

Parental assignment methods

Discussion of results

The theoretical predictions for the number of false parent– offspring pairs, as determined by eqn 29, match very closely to the observed number of false parent–offspring pairs from the simulated data sets. Not surprisingly, the number of false parent–offspring pairs decreases as both number of molecular markers and number of alleles increase. The rate of decrease in false parent–offspring pairs is very similar in data sets with different numbers of loci, but identical allele frequency distributions. Overall, the exclusion probability method presented here predicts the actual number of false parent–offspring pairs with high accuracy and precision. However, even small differences in allele frequencies between adults and juveniles can result in a large overestimation in the number of false parent– offspring pairs when using the approach by Jamieson and Taylor (1997). As shown by Christie (2009), the bias in their method increases with increasing genetic differentiation, whereas Pr(δ) remains unbiased regardless of the level of genetic differentiation.

In practice, however, there may still be occasions when it is better to use allele frequencies from the combined sample of adults and juveniles, such as with small sample sizes or samples with inaccurate allele frequency estimates. It is worthwhile noting that even data sets with 20 loci had some false parent–offspring pairs suggesting that many studies employing strict exclusion may be plagued by false parent–offspring pairs. In fact, this highlights the need for any study employing Mendelian incompatibility to report some measure of exclusionary power. As the theoretical predictions of Pr(δ) and the simulated data match well, this approach can be used confidently to determine how many false parent–offspring pairs are likely to exist in large data sets from natural populations. In particular, our simulations indicate that a minimum number of 18 average molecular markers (with at least 5 alleles each) will be needed in order to obtain a 95% confidence on our parentage assignments.

As pointed out in the present study, parentage assignment may allow for the inference of gene flow and dispersal at ecologically relevant timescales, which has direct implications on the estimate of selection intensity in hybrid zones. However, pedigrees also have an interest in themselves. Despite there has been substantial work on the fate of genes within a given pedigree (e.g. Cannings et al., 1978; Thompson et al., 1978), relatively little work has been done on how

pedigrees themselves evolve. From a genetic point of view, the pedigree constrains what genes can be passed on: with Mendelian inheritance, selection acts solely through the different contributions made by individuals to the pedigree. The recent availability of genomic sequences may focus more attention on pedigrees: given sufficient sequence data, we could infer the pedigree many generations back; and given this pedigree, we could ask what contribution is likely to be made to future generations by each ancestral genome. These questions are long-standing in evolutionary biology, but it will become feasible to answer them only in the next few years.

Parentage Analysis: Limitations of the method

Exclusion is most powerful when there are few candidate parents and highly polymorphic genetic markers available. However, several caveats might be raised when the dataset at hand deviates from this ideal. Even high quality datasets contain errors where at least one allele at a given locus does not match with what we expect from Mendelian laws. Thus it is unwise to exclude a parent immediately when observing such a mismatch. There are many reasons for such mismatches, for example, genotyping errors. Genotyping errors occur when the genotype determined by molecular analysis does not correspond to the real genotype. For instance, common genotyping errors in microsatellite datasets are null alleles, which often result from a mutation in the primer annealing site. The most conservative way to handle a locus with null alleles in parentage analysis is to recode all homozygous genotypes as heterozygotes possessing the detected allele and the null allele, thus preventing exclusion on the basis of homozygous genotypes.

Another potential weakness of a strict exclusion approach is that, if the ages of the sampled individuals are unknown, naïve application of parentage assignment methods can lead to problems where illegal pedigrees are proposed. For example, assigning parentage to a group of individuals from within the same group is likely to result in proposed pedigrees where an individual is its own grandparent. It should also be taken into account that, even though we are not considering extended families (multiple generations mixed together), nor variance in family size (some individuals producing much more offspring than others), these factors would have a similar year-to-year effect than the high immigration levels proposed in the present study, so they will not modify significantly our results. Moreover, despite sampling is done without replacement by default in R (in our simulations we are allowing each adult to produce one descendant only), we could easily define different levels of variance in family size by using a specific probability vector for the offspring sampling process. Finally, it is worth noting that there have been few empirical studies comparing and validating the different approaches, despite the large variety of methodologies that currently exist for addressing many of these problems (e.g. Butler et al., 2004; Csillery et al., 2006; Berger-Wolf et al., 2007).

Future studies: Analysing real data from *Antirrhinum*

Estimating selection from cline width and shape in a real setting

As indicated in the Introduction, the present study was originally motivated by a striking *Antirrhinum* hybrid zone (Whibley et al., 2006). Now we have a theoretical framework that will allow us to answer the questions proposed at the beginning of this report. We can get estimates of natural selection while taking the effect of pollen dispersal (gene flow) into account. From previous studies we know that strong selection acts on several genes coding for the flower color such as SULF, EL and ROS (Whibley et al., 2006). The widths of the clines at these loci, w, relative to the dispersal rate σ^2, will give us a robust estimate of the strength of selection s acting to maintain them through the formula $w \sim \sqrt{\frac{\sigma^2}{2s}}$.

Cline width will be determined directly from the genotyping data using EL, ROS and SULF markers. Note that we will have estimates of dispersal from the parentage studies outlined above.

This is particularly relevant, since estimates of selection derived from cline width are robust because they do not depend strongly on how selection acts, and because equilibrium is reached quickly. In addition, the detailed pattern of genotype frequencies across the hybrid zone can tell us about the form of selection. For example, if alleles are recessive in their effects on fitness, then they will introgress further (Mallet et al., 1990). Similarly, if one recombinant genotype is fitter than other recombinants, then the clines will shift apart in its favour. This kind of asymmetry is what is predicted under Dobzhansky-Muller models of speciation, and provides a route by which populations can diverge to become incompatible with each other, without passing through “adaptive valleys”. A good example is found in the burnet moth, *Zygaena* (Barton et al., 2007). A further issue that can be assessed is whether allele differences involve large shifts along an L-shaped path of high fitness (as in the classic Dobzhansky-Muller model) or whether it involves smaller steps

that take populations along a more diagonal path (Whibley et al., 2006), but those considerations fall well outside the scope of the present study.

Even though so far selection has only been observed for the three flower colour loci, it is possible that other loci are also subject to selection either via differences in pollination niche, or other components of fitness. In fact, hybrid zones that are identified by one selected trait usually also prove to have other traits under selection (Barton and Hewitt, 1985), and one way of inferring selection at other loci is by measuring the barrier to gene flow at the zone of contact. It is worth noting that, although the short scale estimates of gene flow and fitness through parentage analysis have the advantage of being direct, they may be subject to environmental and population fluctuations from year to year, with temporal variation in the magnitude of selection. If selection is less than a few percent such studies will be unable to detect it. Therefore, it is important to complement the above estimates with studies over the medium scale. Patterns of allele frequency at neutral markers can tell us about the cumulative effects of gene flow across the whole genome. The variance in allele frequency over distance, FST, is widely used to estimate the relative rates of gene flow and random drift (Slatkin and Barton, 1990). Where populations are divided into discrete demes, FST allows the value of Nm to be estimated, where N is the deme size and m the number of migrants between demes per generation. Values of FST can be obtained for the same marker loci used for paternity studies and from these estimates we could then infer dispersal rates m. These values could be then compared to those obtained from the short scale parentage studies. This type of integrative analysis of evolution at different temporal scales has never been carried out so far, and it will surely present further challenges for the mathematical treatment of evolutionary processes.

Acknowledgements

Thanks are due to all professors lecturing at MSc INVESTMAT both in Burjassot (Universitat de València) and Tarongers (Universitat Politècnica de València), I really appreciate their effort in making mathematical concepts accessible to a biologist and their understanding with my insistence on bringing biology-related issues into focus when discussing every topic, from Algebra to Fuzzy Topology. Thanks are also due to prof. Guillermo Ayala, who kindly accepted to supervise this work. To all of them my best regards. This work was supported by a MSc fellowship awarded by LaCaixa Foundation to FP. This MSc thesis was carried out in order to fulfill the MSc INVESTMAT requirements.

References

Barton, N.H. (1979). The dynamics of hybrid zones. *Heredity* 43: 341-359.

Barton, N.H. (1983). Multilocus clines. *Evolution* 37: 454-71.

Barton, N.H. and B.O. Bengtsson. (1986). The barrier to genetic exchange between hybridising populations. *Heredity* 57: 357-376.

Barton, N.H. and K.S. Gale. (1993). Genetic analysis of hybrid zones. In: Hybrid zones and the evolutionary process (ed. R.G. Harrison), pp. 13-45. Oxford: Oxford University Press.

Barton, N.H. and G.M. Hewitt. (1985). Analysis of hybrid zones. *Ann. Rev. Ecol. and Syst.* 16: 113-148.

Barton, N H, and M. Turelli. (1991). Natural and sexual selection on many loci. *Genetics* 127: 229–255.

Barton, N.H., Briggs, D.E.G., Eisen, J.A., Goldstein, D.B. and N.H. Patel. (2007). Evolution. Cold Spring Harbor Laboratory Press.

Bazykin, A.D. (1969). Hypothetical mechanism of speciation. *Evolution* 23: 685-731.

Bazykin, A.D. (1972). The disadvantage of heterozygotes in a population within a continuous area. *Genetika* 8: 162-67

Berger-Wolf, T.Y., Sheikh, S.I., DasGupta, B. et al. (2007). Reconstructing sibling relationships in wild populations. *Bioinformatics* 23: 49-56.

Boecklen, W.J. and D.J. Howard. (1997). Genetic analysis of hybrid zones: number of markers and power of resolution. *Ecology* 78: 2611-2616.

Butler, K., Field, C., Herbinger, C.M. and B.R. Smith. (2004). Accuracy, efficiency and robustness of four algorithms

allowing full-sib reconstruction from DNA marker data. *Molecular Ecology* 13: 1589-1600.

Capanna, E. (1980). Chromosomal rearrangement and speciation in progress in *Mus musculus*. *Folia Zoologica* 29: 43-57.

Cannings, C., E. A. Thompson, and M. Skolnick. (1978). Probability functions on complex pedigrees. *Advances in Applied Probability* 10: 26-61.

Chakraborty, R., Meagher, T.R. and P.E. Smouse. (1988). Parentage analysis with genetic markers in natural populations. I. The expected proportion of offspring with unambiguous paternity. *Genetics* 118: 527-536.

Christie, M. (2009). Parentage in natural populations: novel methods to detect parent-offspring pairs in large data sets. *Molecular Ecology Resources* doi: 10.1111/j.1755-0998.2009.02687.x

Csillery, K., Johnson, T., Beraldi, D. et al. (2006). Performance of marker based relatedness estimators in natural populations of outbred vertebrates. *Genetics* 173: 2091-2101.

Derycke, S., Remerie, T., Backeljau, T. et al. (2008). Phylogeography of the *Rhabditis* (*Pellioditis*) marina species complex: evidence for long-distance dispersal, and for range expansions and restricted gene flow in the northeast Atlantic. *Molecular Ecology* 17: 3306-3322.

Devlin, B. et al. (1988). Fractional paternity assignment: theoretical development and comparison to other methods. *Theoretical and Applied Genetics* 76: 369-380.

Dodds, K.G., Tate, M.L., McEwan, J.C., Crawford, A.M. (1996). Exclusion probabilities for pedigree testing farm animals. *Theoretical and Applied Genetics* 92: 966-975.

Edwards, A.W.F. (1968). Simulation studies of genealogies. *Heredity* **23**: 628.

Felsenstein, J. (1974). The evolutionary advantage of recombination. *Genetics* 78: 737-756.

Felsenstein, J. (1976). The theoretical population genetics of variable selection and migration. *Ann. Rev. Genet.* 10: 253-80.

Fisher, R.A. (1930). The Genetical Theory of Natural Selection.Oxford: Oxford University Press.

Fisher, R.A. (1937). The wave of advance of advantageous genes. *Ann. Eugenics* 7: 355-369.

Garber, R.A. and J.W. Morris. (1983). General equations for the average power of exclusion for genetic systems of n codominant alleles in one-parent and in no-parent cases of disputed parentage. In: Inclusion Probabilities in Parentage Testing (ed. R.H. Walker), pp. 277–280. American Association of Blood Banks, Arlington, VA.

Haldane, J.B.S. (1948). The theory of a cline. *Journal of Genetics* 48: 277-84.

Harrison, R.G. (1993). Hybrid zones and the evolutionary process. Oxford: Oxford University Press.

Hill, W.G. and A. Robertson. (1966). The effect of linkage on limits to artificial selection. *Genetics Research* 8: 269-294.

Jamieson, A. and S.S. Taylor. (1997). Comparisons of three probability formulae for parentage exclusion. *Animal Genetics* 28: 397-400.

Kauserud, H., Stensrud, O., Decock, C., Shalchian-Tabrizi, K. and T. Schumacher. (2006). Multiple gene genealogies and AFLPs suggest cryptic speciation and long-distance dispersal in the basidiomycete *Serpula himantioides* (Boletales). *Molecular Ecology* 15: 421–431.

Kirkpatrick, M., Johnson, T. and N.H. Barton (2002). General models of multilocus evolution. *Genetics* 162: 1727-1750.

Jones, A.G. and W.R. Ardren. (2003). Methods of parentage analysis in natural populations. *Molecular Ecology* 12: 2511-2523.

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

Li, W.H. and M. Nei. (1974). Stable linkage disequilibrium without epistasis in subdivided populations. *Theoretical Population Biology* 6: 173-183.

Prout, T. (1973) Epistatis between functionally related isoenzymes of *Mytilus edulis*. *Genetics* 73: 493-496.

Manel, S., Gaggiotti, O.E. and R.S. Waples. (2005). Assignment methods: matching biological questions with

appropriate techniques. *Trends in Ecology and Evolution* 20: 136-142.

MacCluer, J.W., Vandeberg, J.L., Read, B. and O.A. Ryder. (1986). Pedigree analysis by computer simulation. *Zool Biol* 5: 147-160.

Mallet J, Barton N, Lamas G, Santisteban J, Muedas M, Eeley H. (1990). Estimates of selection and gene flow from measures of cline width and linkage disequilibrium in Heliconius hybrid zones. Genetics 124: 921-936.

Milligan, B.G. (2003). Maximum-likelihood estimation of relatedness. *Genetics* 163: 1153-1167.

Moore, W.S. (1977). An evaluation of narrow hybrid zones in vertebrates. *Q. Rev. Biol.* 52: 263-78.

Nagylaki, T. (1975). Conditions for the existence of clines. Genetics 80, 595-615.

Nason, J. D. and N. C. Ellstrand. (1993). Estimating the frequencies of genetically distinct classes of individuals in hybridized populations. Journal of Heredity 84: 1-12.

Nathan R (2006). Long-distance dispersal of plants. Science, 313: 786–788.

Nielsen R, Mattila DK, Clapham PJ, Palsboll PJ (2001). Statistical approaches to paternity analysis in natural populations and applications to the North Atlantic humpback whale. Genetics 157: 1673-1682.

Queller DC, Zacchi F, Cervo R et al. (2000). Unrelated helpers in a social insect. Nature. 405: 784–787.

Rouhani, S. and N. H. Barton. (1987). Speciation and the "shifting balance" in a continuous population. *Theoretical Population Biology* 31: 465-492.

Slatkin, M. (1973). Gene flow and selection in a cline. *Genetics* 75: 733-56.

Slatkin, M. (1975). Gene flow and selection in a two-locus system. *Genetics* 81: 209-22.

Slatkin, M. and N.H. Barton. (1990). A comparison of three methods for estimating average levels of gene flow. *Evolution* 43: 1349-1368.

Thompson, E.A., C. Cannings and M.H. Skolnick. (1978). Ancestral inference I. The problem and the method. *Annals of Human Genetics* 42: 95-108.

Turelli, M. and N.H. Barton. (1994). Genetic and statistical analyses of strong selection on polygenic traits: what, me normal? *Genetics* 138: 913-941.

Whibley, A.C., Langlade, N.B., Andalo, C., Hanna, A.I., Bangham, A., Thebaud, C. and E. Coen. (2006). Evolutionary paths underlying flower color variation in Antirrhinum. *Science* 313: 963-966.

Wolfram, S. (1996). The Mathematica Book, 3rd edition. Wolfram Media, Cambridge Univ. Press, Cambridge.

APPENDIX I: CLINE SIMULATION AND PEDIGREE FUNCTION DEFINITION

AlleleFrequencies

```
AlleleFrequencies::usage =
  "AlleleFrequencies[{p1,p2,…},n] represents allele frequencies,
    and deme size n.  AlleleFrequencies[{p1,p2,…}] is shorthand
    for AlleleFrequencies[{p1,p2,…},1]";

AlleleFrequencies[ψ_List] := AlleleFrequencies[ψ, 1];
```

AlleleFrequencyMean

```
AlleleFrequencyMean::usage =
  "AlleleFrequencyMean[ψ] gives the average frequency of 1
    alleles for the population representation ψ.  Also applies
    to a list of ψ's, representing a cline.  For representations
    SymmetricNeutralHaploidFrequencies or
    SymmetricNeutralDiploidFrequencies, returns {p,ū}, where
    ū is the average neutral allele frequency";

AlleleFrequencyMean[ψ : {PopulationRepresentations[_, _] ...},
   opts___Rule] := AlleleFrequencyMean[#, opts] & /@ ψ;
AlleleFrequencyMean[z : PopulationRepresentations[_, _], opts___Rule] :=
  AlleleFrequencyTable[z, opts] /.
   {{x_List, y_} :> {Mean[x], y}, x_List :> Mean[x]};
```

AlleleFrequencyTable

```
AlleleFrequencyTable::usage =
  "AlleleFrequencyTable[ψ] gives the frequency of 1 alleles for
    each locus.  Also applies to a list of ψ's, representing
    a cline. For representations SymmetricNeutralHaploidFrequencies
    or SymmetricNeutralDiploidFrequencies, returns {{p,p,…},ū},
    where ū is the average neutral allele frequency.";
```

```
AlleleFrequencyTable[ψ : {PopulationRepresentations[_, _] ...},
   opts___Rule] := AlleleFrequencyTable[#, opts] & /@ ψ;
AlleleFrequencyTable[AlleleFrequencies[p_, _], ___Rule] := p;
AlleleFrequencyTable[ψ : DiploidRepresentations[_, _], opts___Rule] :=
  AlleleFrequencyTable[MakeHaploid[ψ], opts];
AlleleFrequencyTable[HaploidFrequencies[ψ_, _], opts___Rule] :=
  Plus @@ (HaploidTypes[Log[2, Length[ψ]], opts] ψ);
AlleleFrequencyTable[SymmetricHaploidFrequencies[ψ_, _], opts___Rule] :=
  Module[{n = Length[ψ] - 1, p},
   If[n == 0, Indeterminate, p = ψ.Range[0, 1, 1 / n]; Table[p, {n}]]];
AlleleFrequencyTable[SymmetricNeutralHaploidFrequencies[{ψ_, u_}, _],
  opts___Rule] := Module[{n = Length[ψ] - 1, pb, ub},
  If[n == 0, {Indeterminate, Indeterminate}, pb = ψ.Range[0, 1, 1 / n];
   ub = ψ.u; {Table[pb, {n}], ub}]];
AlleleFrequencyTable[HaploidIndividuals[ψ_, n_], opts___Rule] :=
 (Plus @@ ψ) / n;
```

BarrierStrength

```
BarrierStrength::usage =
  "BarrierStrength[u] gives the barrier strengths {Δu/∂ₓu₋,Δu/∂ₓu₊},
    based on the list of allele frequencies u₁…uₙ.  The default
    is to assume that u₀=u₁, SubscriptBox[u,n,  1]=uₙ.
    FixedEnds→{u₀,SubscriptBox[u,n,  1]} sets the outer demes
    to have allele frequencies u₀,SubscriptBox[u,n,  1]
    instead; sensible results are only obtained with this
    option set, since otherwise the gradients at the edges
    are zero. Also works with pairs {p,u}, as derived under
    SymmetricNeutralHaploidFrequencies or
    SymmetricNeutralDiploidFrequencies. ";

Options[BarrierStrength] = {FixedEnds → {{}, {}}};

BarrierStrength[u_, opts___] :=
    Module[{uu = u, gr = ClineGradient[u, opts], du, ps},
    If[TensorRank[gr] == 2,
            uu = Transpose[uu] // Last;
            gr = Transpose[gr] // Last];
ps = Position[gr, Max[gr]][[1, 1]] - 0.5;
du = (Last[uu] - Last[gr] (Length[uu] - ps)) -
     (First[uu] + First[gr] (ps - 1));
du / {First[gr], Last[gr]}];
```

ClineGradient

```
ClineGradient::usage =
  "ClineGradient[p] gives the gradient (pᵢ-SubscriptBox[p,i -
    1]) of the list of allele frequencies p₁…pₙ; returns the
    list {p₁-p₀, SubscriptBox[…p,n, 1]-pₙ}.  The default is
    to assume that p₀=p₁, SubscriptBox[p,n, 1]=pₙ.
    FixedEnds→{p₀,SubscriptBox[p,n, 1]} sets the outer demes
    to have allele frequencies p₀,SubscriptBox[p,n, 1]  instead.";
```

```
Options[ClineGradient] = {FixedEnds → {{}, {}}};

ClineGradient[p_, opts___] :=
  Module[{pp, fe = FixedEnds /. {opts} /. Options[ClineGradient]},
        pp = If[(fe // Flatten // Length) == 0,
                Prepend[Append[p, Last[p]], First[p]],
                Prepend[Append[p, Last[fe]], First[fe]]];
        Drop[RotateLeft[pp] - pp, -1]];
```

ClineWidth

```
ClineWidth::usage =
  "ClineWidth[{p0,…}] gives the cline width, defined as the
    maximum gradient  in allele frequencies, pi.  ClineWidth[ψ],
    where ψ is a list of population representations, lists
    the cline widths for each locus. For representations
    SymmetricNeutralHaploidFrequencies or
    SymmetricNeutralDiploidFrequencies, ClineWidth[ψ] returns
    the widths of the selected and the neutral clines.";

ClineWidth::badDepth =
  "The cline `1` does not have an appropriate structure.";

ClineWidthRaw[p_List] := 1 / Max[Drop[p - RotateRight[p], 1]];
ClineWidth[ψ : {PopulationRepresentations[_, _] ...}, opts___] :=
  Module[{p = AlleleFrequencyTable[ψ, opts]},
   If[Length[p] < 2, Indeterminate,
    Switch[TensorRank[p],
     2, ClineWidthRaw /@ Transpose[p],
      3, {ClineWidthRaw /@ Transpose[First /@ p],
      ClineWidthRaw /@ Transpose[Last /@ p]},
       _, Message[ClineWidth::badDepth, ψ]; Indeterminate]]];
```

ConstructMutateList

```
ConstructMutateList::usage =
  "ConstructMutateList[{{μ}}] converts shorthand versions of a
    list of mutation rates into the full form, {{μ,μ},…}";

ConstructMutateList[ml_, n_Integer] := Module[{mut},
   mut = ml /. {μ_?(Not[ListQ[#]] &)} → {μ, μ};
   If[Depth[mut] == 2, Array[mut &, n], mut]];
```

ConstructMutateMatrix

```
ConstructMutateMatrix::usage =
  "ConstructMutateMatrix[{{μ,ν},…},ngenes,NumericalModel->
    HaploidFrequencies] generates the matrix which
    determines the effects of mutation on haploid genotype
    frequencies. NumericalModel->SymmetricFrequencies or
    SymmetricNeutralFrequencies can be used.  Compiled->True
    is faster for numerical values.";
```

```
Options[ConstructMutateMatrix] =
  {Compiled → False, NumericalModel → HaploidFrequencies};

ConstructMutateMatrix::badModel =
  "`1` is not a valid setting for NumericalModel in
    ConstructMutateMatrix";

ConstructMutateMatrix[mr_List, n_Integer, opts___Rule] :=
  ConstructMutateMatrix[mr, n, opts] =
   Module[{nm = NumericalModel /. {opts} /. Options[ConstructMutateMatrix],
     tmr, μ, ν, ht, i, j, a},
    If[Compiled /. {opts} /. Options[ConstructMutateMatrix],
     Switch[nm,
      HaploidFrequencies,
      ConstructMutateMatrixCompiled[nm][mr, HaploidTypes[n, opts]],
      SymmetricHaploidFrequencies | SymmetricNeutralHaploidFrequencies,
      ConstructMutateMatrixCompiled[nm][mr, n],
      _,
      Message[ConstructMutateMatrix::badModel, nm];],
     Switch[nm,
      HaploidFrequencies,
      tmr = Transpose[mr]; ht = HaploidTypes[n, opts];
      μ = tmr[[1]]; ν = tmr[[2]];
      Outer[Times @@ ((1 - μ) - #1 (1 - 2 μ) - #2 (1 - 2 #1) (1 - μ - ν)) &,
       ht, ht, 1],
      SymmetricHaploidFrequencies,
      μ = mr[[1]]; ν = mr[[2]]; a = (μ/(1 - μ)  ν/(1 - ν));
      Table[If[i > j,
        μ^(i-j) (1 - μ)^(n-i) (1 - ν)^j Binomial[n - j, n - i]
         Hypergeometric2F1[-j, i - n, 1 + i - j, a],
        If[i == j,
         (1 - μ)^(n-j) (1 - ν)^j Hypergeometric2F1[-j, j - n, 1, a],
         (1 - μ)^(n-j) (1 - ν)^i ν^(j-i) Binomial[j, i]
          Hypergeometric2F1[-i, j - n, 1 - i + j, a]]],
       {i, 0, n}, {j, 0, n}],
      SymmetricNeutralHaploidFrequencies,
      0,
      _,
      Message[ConstructMutateMatrix::badModel, nm];]]];
```

ConstructMutateMatrixCompiled

```
ConstructMutateMatrixCompiled::usage =
  "ConstructMutateMatrixCompiled[HaploidFrequencies][{{μ,ν},…},{{0,…},…}]
     gives the mutation matrix; called by
    ConstructMutateMatrix[…,Compiled->True].
    ConstructMutateMatrixCompiled[SymmetricHaploidFrequencies][{μ,ν},
    ngenes] is also defined, as is
    ConstructMutateMatrixCompiled[SymmetricHaploidFrequencies][{μ,ν},
    ngenes].";

ConstructMutateMatrixCompiled[HaploidFrequencies] =
  Compile[{{mr, _Real, 2}, {ht, _Integer, 2}},
   Module[{tmr = Transpose[mr], j, k},
    Table[
     Times @@ ((1 - tmr[[1]]) - ht[[j]] (1 - 2 tmr[[1]]) -
        ht[[k]] (1 - 2 ht[[j]]) (1 - tmr[[1]] - tmr[[2]])), {j, Length[ht]},
     {k, Length[ht]}]],
   {{tmr, _Real, 2}, {j, _Integer}, {k, _Integer}}];

ConstructMutateMatrixCompiled[SymmetricHaploidFrequencies] =
  Compile[{{mr, _Real, 1}, {n, _Integer}},
   Module[{i, j, a, μ, ν},
    μ = mr[[1]]; ν = mr[[2]]; a = (μ/(1 - μ)) (ν/(1 - ν));
    Table[If[i > j,
      μ^(i-j) (1 - μ)^(n-i) (1 - ν)^j Binomial[n - j, n - i]
       Hypergeometric2F1[-j, i - n, 1 + i - j, a],
      If[j == i,
       (1 - μ)^(n-j) (1 - ν)^j Hypergeometric2F1[-j, j - n, 1, a],
       Binomial[j, i] (1 - μ)^(n-j) (1 - ν)^i ν^(j-i)
        Hypergeometric2F1[-i, j - n, 1 - i + j, a]]],
     {i, 0, n}, {j, 0, n}]],
   {{a, _Real}, {μ, _Real}, {ν, _Real}, {i, _Integer}, {j, _Integer},
    {Binomial[_, _], _Real}}];
```

DemeSize

```
DemeSize::usage =
  "DemeSize[ψ] gives the deme size associated with the population
    representation ψ; also applies to a cline.";

DemeSize[PopulationRepresentations[_, n_]] := n;
DemeSize[ψ : {PopulationRepresentations[_, _] ...}] := DemeSize /@ ψ;
```

DiploidFrequencies

```
DiploidFrequencies::usage =
  "DiploidFrequencies[{{SubscriptBox[ψ,000, 000],…},…},n]
    represents diploid genotype frequencies, and deme size
    n.  DiploidFrequencies[{{SubscriptBox[ψ,000, 000],…},…}]
    is shorthand for DiploidFrequencies[{{SubscriptBox[ψ,000,
    000],…},…},1]";

DiploidFrequencies[ψ_List] :=
  DiploidFrequencies[ψ, 1];
```

DiploidIndividuals

```
DiploidIndividuals::usage =
  "DiploidIndividuals[{{{0,0,1…},{0,1,1},…}},…},n] represents
    a population of n diploid individuals; n must match the
    length of the list, and is added automatically if
    DiploidIndividuals[{{{0,0,1…},{0,1,1},…}},…}] is specified.";

DiploidIndividuals[ψ_List] := DiploidIndividuals[ψ, Length[ψ]];
```

DiploidRepresentations

```
DiploidRepresentations::usage =
  "DiploidRepresentations is a pattern which matches any valid
    representation of a diploid population.";

DiploidRepresentations = DiploidFrequencies | AlleleFrequencies |
   SymmetricDiploidFrequencies | SymmetricNeutralDiploidFrequencies |
   DiploidIndividuals;
```

DisequilibriumMean

```
DisequilibriumMean::usage =
  "DisequilibriumMean[ψ] gives the mean pairwise linkage
    disequilibrium for the population representation ψ.  Also
    applies to a list of ψ's, representing a cline.  For
    representations SymmetricNeutralHaploidFrequencies or
    SymmetricNeutralDiploidFrequencies, returns the mean D
    between selected loci, and the mean D between the neutral
    and selected loci.  ";
```

```
DisequilibriumMean[ψ : {PopulationRepresentations[_, _] ...},
   opts___Rule] :=
  DisequilibriumMean[#, opts] & /@ ψ;
DisequilibriumMean[pr : NonNeutralRepresentations[_, _], opts___Rule] :=
 Module[{n, dt = DisequilibriumTable[pr, opts]},
  n = Length[dt];
  Plus @@ Plus @@ ((Array[1 &, {n, n}] - IdentityMatrix[n]) dt) 1/(n (n - 1))];
DisequilibriumMean[pr : NeutralRepresentations[_, _], opts___Rule] :=
 Module[{n, dt = DisequilibriumTable[pr, opts]},
  n = Length[dt[[1]]];
  {Plus @@ Plus @@ ((Array[1 &, {n, n}] - IdentityMatrix[n]) dt[[1]]) 1/(n (n - 1)),
   Mean[dt[[2]]]}];
```

DisequilibriumTable

```
DisequilibriumTable::usage =
  "DisequilibriumTable[ψ] gives a matrix of pairwise linkage
    disequilibrium for the population representation ψ.  Also
    applies to a list of ψ's, representing a cline.  For
    representations SymmetricNeutralHaploidFrequencies or
    SymmetricNeutralDiploidFrequencies, returns the D between
    selected loci, and the D between the neutral and selected
    loci. DisequilibriumMean[ψ] gives the mean pairwise linkage
    disequilibrium for the population representation ψ.  Also
    applies to a list of ψ's, representing a cline.  For
    representations SymmetricNeutralHaploidFrequencies or
    SymmetricNeutralDiploidFrequencies, returns the mean D
    between selected loci, and the mean D between the neutral
    and selected loci.  ";
```

```
DisequilibriumTable[ψ : {PopulationRepresentations[_, _] ...},
   opts___Rule] :=
  DisequilibriumTable[#, opts] & /@ ψ;
DisequilibriumTable[AlleleFrequencies[p_, _], ___Rule] :=
  DiagonalMatrix[p (1 - p)];
DisequilibriumTable[ψ : DiploidRepresentations[_, _], opts___Rule] :=
  DisequilibriumTable[MakeHaploid[ψ], opts];
DisequilibriumTable[HaploidFrequencies[ψ_, _], opts___Rule] :=
  Module[{n = Log[2, Length[ψ]],
    p = AlleleFrequencyTable[HaploidFrequencies[ψ], opts], dk},
    dk = (# - p) & /@ HaploidTypes[n, opts];
    ψ.(Outer[Times, #, #] & /@ dk)];
DisequilibriumTable[SymmetricHaploidFrequencies[ψ_, nd_],
   opts___Rule] :=
  Module[{n = Length[ψ] - 1,
    p = AlleleFrequencyMean[SymmetricHaploidFrequencies[ψ, nd], opts],
    d, du},
   d = (ψ.(Range[0, n] - n p)^2 - n p (1 - p))/(n (n - 1));
   IdentityMatrix[n] (p (1 - p) - d) + Array[d &, {n, n}]];
DisequilibriumTable[SymmetricNeutralHaploidFrequencies[{ψ_, u_}, nd_],
   opts___Rule] :=
  Module[{n = Length[ψ] - 1,
    pu = AlleleFrequencyTable[SymmetricNeutralHaploidFrequencies[
        {ψ, u}, nd], opts], p, ub, d, du},
   p = pu[[1, 1]]; ub = pu[[2]];
   d = (n ψ.(Range[0, 1, 1/n] - p)^2 - p (1 - p))/(n - 1);
   du = (Range[0, 1, 1/n] - p) ψ).(u - ub);
   {IdentityMatrix[n] (p (1 - p) - d) + Array[d &, {n, n}],
    Array[du &, {n}]}];
DisequilibriumTable[HaploidIndividuals[ψ_, nd_], opts___Rule] :=
  Module[{nl = Length[ψ[[1]]],
    p = AlleleFrequencyTable[HaploidIndividuals[ψ, nd]]},
    If[nd < 2, Array[Indeterminate &, {nl, nl}],
    Plus @@ (Outer[Times, # - p, # - p] & /@ ψ) / (nd - 1)]];
```

DoubleCline

```
DoubleCline::usage =
  "DoubleCline[ψ,EvenQ[ndemes]] takes the cline ψ to represent
    the left half of a symmetric cline, and adds the right
    half.  If the total number of demes is even, then the
    second argument is set True, and the extra demes are a
    mirror image of ψ.  If the total number of demes is odd,
    the second argument is set False, and Last[ψ] is assumed
    to be the central deme; it must contain a symmetrical set
    of genotype frequencies. ";

DoubleCline[ψ : {(IndividualRepresentations | AlleleFrequencies)[_, _] ...},
   eQ_] :=
  Join[ψ,
   Reverse[If[eQ, ψ, Drop[ψ, -1]] /.
     (pr : PopulationRepresentations)[f_, n_] :> pr[1 - f, n]]];

DoubleCline[
   ψ :
    {(HaploidFrequencies | DiploidFrequencies |
         SymmetricHaploidFrequencies | SymmetricDiploidFrequencies)[
       _, _] ...}, eQ_] :=
  Join[ψ,
   Reverse[If[eQ, ψ, Drop[ψ, -1]] /.
     (pr : PopulationRepresentations)[f_, n_] :> pr[Reverse[f], n]]];

DoubleCline[ψ : {NeutralRepresentations[_, _] ...}, eQ_] :=
  Join[ψ,
   Reverse[If[eQ, ψ, Drop[ψ, -1]] /.
     (pr : NeutralRepresentations)[{f_, u_}, n_] :>
      pr[{Reverse[f], 1 - Reverse[u]}, n]]];
```

DropGenes

```
DropGenes::usage =
  "DropGenes[p,P] generates random genotypes, given the pedigree
    P; uses Mendel[]. Mendel[{{0,1,…},{1,0,…}}] gives a random
    gamete, assuming no linkage. DropGenes[p0,{P1,…}] does
    the same for many generations";

DropGenes[p : {{___} ...}, P_SparseArray] := Mendel[p[[#]] & /@ Parents[P];

DropGenes[p : {{___} ...}, P_SparseArray] := Mendel[p[[#]] & /@ Parents[P];

DropGenes[p0 : {{___} ...}, Pl : {___SparseArray}] :=
  FoldList[DropGenes, p0, Pl];
```

ExponentialF

```
ExponentialF::usage =
  "ExponentialF[r,β,λ][n,w̄] is the function rn w̄^β Exp[-λn].
    DemeSize->ExponentialF[r,β,λ] can be passed to compiled
    versions of IterateExact and StoreExact.
    ExponentialF[r,β,γ,λ][n,w̄] represents the generalised
    form rSuperscriptBox[n,1 - γ]w̄^β Exp[-λn].
    ExponentialF[r…][j][n,w̄] is the same as ExponentialF[r…][n,w̄].";

ExponentialF[r_, β_, λ_][n_, wb_] := r n wb^β Exp[-λn];
ExponentialF[r_, β_, γ_, λ_][n_, wb_] := r n^(1-γ) wb^β Exp[-λn];

ExponentialF[r_, β_, λ_][_][n_, wb_] := r n wb^β Exp[-λn];
ExponentialF[r_, β_, γ_, λ_][_][n_, wb_] := r n^(1-γ) wb^β Exp[-λn];
```

FitnessTable

```
FitnessTable::usage =
  "FitnessTable[W,n] gives a table of all possible fitnesses,
    using the fitness function W, for n loci.  NumericalModel
    specifies which representation is used.  Useful for
    tabulating fitness in order to speed up functions such
    as MeanW.  Note that fitnesses cannot be tabulated if
    AlleleFrequencies are used.";

Options[FitnessTable] = {NumericalModel → HaploidFrequencies};

FitnessTable::badOption =
  "Fitness cannot be tabulated for AlleleFrequencies.  A function
    which supplies the mean fitness and the selection
    coefficients as a function of allele frequencies must be used.";

FitnessTable[W_, n_Integer, opts___Rule] :=
  Module[
   {i, j, h, ht, nm = NumericalModel /. {opts} /. Options[FitnessTable]},
        Switch[nm,
    HaploidFrequencies | HaploidIndividuals,
      W /@ HaploidTypes[n, opts],
    DiploidFrequencies | DiploidIndividuals,
      ht = HaploidTypes[n, opts]; Outer[W, ht, ht, 1],
    SymmetricHaploidFrequencies | SymmetricNeutralHaploidFrequencies,
      Table[W[i], {i, 0, n}],
    SymmetricDiploidFrequencies | SymmetricNeutralDiploidFrequencies,
      Table[SymmetricMeanW[i, j, n, W], {i, 0, n}, {j, 0, n}],
    AlleleFrequencies,
     Message[FitnessTable::badOption]; {},
    _,
      Message[NumericalModel::badSetting, nm]; {}]];
```

FixedEnds

```
FixedEnds::usage =
  "FixedEnds is an option for MigrateExact and related functions
    which specifies that endpoints should be fixed.  FixedEnds→{ψ}
    specifies a source population that provides migrants into
    a single deme.  FixedEnds→{ψ0,ψ1} fixes genotype frequencies
    at ψ0, ψ1 at either end of a cline.";
```

FrequencyRepresentations

```
FrequencyRepresentations::usage =
  "FrequencyRepresentations is a pattern which matches any valid
    representation of a population by allele, class or genotype
    frequencies.";

FrequencyRepresentations =
  HaploidFrequencies | DiploidFrequencies | AlleleFrequencies |
   SymmetricHaploidFrequencies | SymmetricDiploidFrequencies |
   SymmetricNeutralHaploidFrequencies |
   SymmetricNeutralDiploidFrequencies;
```

Gametes

```
Gametes::usage =
  "Gametes[Y,Z] gives the full distribution of 2^noffspring
    genotypes. Lligam→{SubscriptBox[\(r\),\(1,\(\(\\)\(
    \)\(\(\(\\)\( \)\)\(\(\(\(\\)\( \)\)\)\(\ \(\(\(\(\\)\(
    \)\)\)\) \(\(\(\(\\)\( \)\)\)\) \(\(\(\(\\)\( \)\)\)\)\
    2\)\)\)\)\)],SubscriptBox[\(r\),\(2,\(\(\\)\( \)\(\(\(\\)\(
    \)\)\(\(\(\(\\)\( \)\)\)3\)\)\)\)],…} specifies a linear
    genetic map. Gametes[X,Y,Z] is also defined, and gives
    the probability of a specific gamete X from diploid parent
    {Y,Z}. Gametes[i,j,n,NumericalModel→SymmetricHaploidFrequencies]
    gives the distribution of genotypes among gametes from a
    parent of genotype {i,j}.
    Gametes[i,j,n,W,NumericalModel→SymmetricHaploidFrequencies]
    gives the same, for fitness W[i,j,h]. NOT normalised by
    MeanW[], however.";

Options[Gametes] = {Lligam → False};

Gametes[i_Integer, j_Integer, n_Integer] := Gametes[i, j, n, 1 &];
Gametes[Y_List, Z_List, opts___Rule] := Gametes[Y, Z, 1 &, opts];

Gametes[i_Integer, j_Integer, n_Integer, W_] :=
  Module[{l, k},
    2^-(i+j) / Binomial[n, j]
    Sum[2^2 l Table[Binomial[i + j - 2 l, k - l], {k, 0, n}] Binomial[i, l]
      Binomial[n - i, j - l] W[i, j, i + j - 2 l],
     {l, Max[0, i + j - n], Min[i, j]}]];
```

```
Gametes[Y_List, Z_List, W_, opts___Rule] :=
  (GametesRaw[#, Y, Z, opts] & /@ HaploidTypes[Length[Z], opts]) W[Y, Z];

GametesRaw[X_List, Y_List, Z_List, opts___Rule] :=
    Module[{ph, rr, rQ,
            r = Lligam /. {opts} /. Options[Gametes],
            h = Y + Z - 2 Y Z,
            p = (1 - X - Y + 2 X Y)},
      ph = Flatten[Position[h, 1]];
   If[Times @@ (h + (1 - h) p) == 0,
            0,
            If[(r === False) || ph == {},
              Times @@ (1 - h / 2),
           rr = Prepend[NetRecombination /@

        (Take[r, #] & /@ Drop[Transpose[{ph, RotateLeft[ph - 1]}], -1]),
       1 / 2];
           rQ = p[[#]] & /@ ph;
                 rQ = Abs[rQ - RotateRight[rQ]];
            Times @@ ((1 - rr) + rQ (2 rr - 1))]]];
```

GameteTable

```
GameteTable::usage =
  "GameteTable[n] gives the full distribution of 2^n  offspring
    genotypes. Lligam→{SubscriptBox[\(r\),\(1,\(\(\\)\(
    \)\(\(\(\\)\( \)\)\(\(\(\(\\)\( \)\)\)\(\(\(\\)\(
    \)\)\)2\)\)\)\)],SubscriptBox[\(r\),\(2,\(\(\\)\(
    \)\(\(\(\\)\( \)\)\(\(\(\(\\)\( \)\)\)\(\(\(\\)\(
    \)\)\)3\)\)\)\)],…} specifies a linear genetic map.
    GameteTable[n,NumericalModel→SymmetricHaploidFrequencies]
    stores a matrix showing the distribution of genotypes
    among gametes, for each diploid genotype, under the
    symmetric model. GameteTable[n,W] is also defined; it
    allows for a fitness function W.   Results are stored,
    which greatly speeds calculations.  However, if fitnesses
    continually change, the option StoreResults→False must be used.";

Options[GameteTable] = {Lligam → False, StoreResults → True,
   NumericalModel → HaploidFrequencies};

GameteTable::badOpts = "`1` is not a valid option for GameteTable";

GameteTable[n_Integer, opts___Rule] := GameteTable[n, 1 &, opts];
```

```
GameteTable[n_Integer, ww_, opts___Rule] :=
  Module[
   {i, j, ht,
    nm = NumericalModel /. {opts} /. Options[GameteTable],
    srQ = StoreResults /. {opts} /. Options[GameteTable]},
   If[srQ,
       GameteTable[n, ww, opts] =
     GameteTable[n, ww, StoreResults → False, opts],
    Switch[nm,
     SymmetricRepresentations,
     Table[Gametes[i, j, n, ww], {i, 0, n}, {j, 0, n}],
     NonSymmetricRepresentations,
     ht = HaploidTypes[n, opts];
         Outer[Gametes[#1, #2, ww, opts] &, ht, ht, 1],
     _,
     Message[GameteTable::badOpts, nm]]]];
```

HaploidFrequencies

```
HaploidFrequencies::usage =
  "HaploidFrequencies[{ψ000,…},n] represents haploid genotype
    frequencies, and deme size n.  HaploidFrequencies[{ψ000,…}]
    is shorthand for HaploidFrequencies[{ψ000,…},1]";

HaploidFrequencies[ψ_List] :=
  HaploidFrequencies[ψ, 1];
```

HaploidIndividuals

```
HaploidIndividuals::usage =
  "HaploidIndividuals[{{0,0,1…},…},n] represents a population
    of n haploid individuals; n must match the length of the
    list, and is added automatically if
    HaploidIndividuals[{{0,0,1…},…}] is specified.";

HaploidIndividuals[ψ_List] := HaploidIndividuals[ψ, Length[ψ]];
```

HaploidRepresentations

```
HaploidRepresentations::usage =
  "HaploidRepresentations is a pattern which matches any valid
    representation of a haploid population.";

HaploidRepresentations = HaploidFrequencies | AlleleFrequencies |
   SymmetricHaploidFrequencies | SymmetricNeutralHaploidFrequencies |
   HaploidIndividuals;
```

- **HaploidTypes[{{a1,b1,..},{a2,b2,..}..}] lists all possible haploid genotypes, based on the set of alleles {a1,b1,..} at locus 1, etc. HaploidTypes[{{a,b,..},n}] lists all possible genotypes, assuming the same set of alleles at n loci. HaploidTypes[n] assumes n loci, with two alleles labelled {0,1}. HaploidTypes[n,SortByClass→True] lists haplotypes ordered by the # of '1' alleles.**

```
HaploidTypes::usage =
  "HaploidTypes[{{a1,b1,..},{a2,b2,..}..}] lists all possible
    haploid genotypes, based on the set of alleles {a1,b1,..}
    at locus 1, etc.  HaploidTypes[{{a,b,..},n}] lists all
    possible genotypes, assuming the same set of alleles at
    n loci.  HaploidTypes[n] assumes n loci, with two alleles
    labelled {0,1}. HaploidTypes[n,SortByClass→True] lists
    haplotypes ordered by the # of '1' alleles.";

Options[HaploidTypes] = {SortByClass → False};

HaploidTypes[{}, ___] := {};

HaploidTypes[g_List, opts___] :=
  HaploidTypes[g, opts] =
   If[SortByClass /. {opts} /. Options[HaploidTypes],
            Sort[HaploidTypes[g],
     Module[{p1 = Plus @@ #1, p2 = Plus @@ #2},
       If[p1 < p2, True, If[p1 > p2, False, OrderedQ[{#1, #2}]]]] &],
            Flatten[Apply[Outer, Prepend[g, List]], Length[g] - 1]];

HaploidTypes[{g_List, n_Integer}, opts___] :=
  HaploidTypes[{g, n}, opts] = HaploidTypes[Table[g, {n}], opts];

HaploidTypes[n_Integer, opts___] :=
  HaploidTypes[n, opts] = HaploidTypes[{{0, 1}, n}, opts];
```

- **HeterozygoteB**

```
HeterozygoteB::usage =
  "HeterozygoteB[i,j,n] gives the distribution of the # of
    heterozygous loci among gametes from a parent of genotype
    {i,j}, under the symmetric model.  HeterozygoteB[n] stores
    a table of HeterozygoteB[i,j,n] for i=0…n, j=0…n.";

HeterozygoteB[i_, j_, n_] := HeterozygoteB[i, j, n] =
   Module[{l, pp = Array[0 &, n + 1]},
            Do[pp[[i + j - 2 l + 1]] = Binomial[i, l] Binomial[n - i, j - l ],
     {l, Max[0, i + j - n], Min[i, j]}]; pp / Binomial[n, j]];

HeterozygoteB[n_] := HeterozygoteB[n] =
        Module[{i, j}, Table[HeterozygoteB[i, j, n], {i, 0, n},
     {j, 0, n}]];
```

IndividualRepresentations

```
IndividualRepresentations::usage =
  "IndividualRepresentations is a pattern which matches any
    valid representation of a population as a list of individuals.";

IndividualRepresentations = HaploidIndividuals | DiploidIndividuals;
```

IterateExact

```
IterateExact::usage =
  "IterateExact[ψ,W,0] gives gamete frequencies after one
    generation of selection; ψ may represent a single deme,
    or an array of demes, but must represent a haploid
    population. IterateExact[ψ,W,m] allows for migration.
    If ψ contains only a single population, then it iterates
    that deme, with immigration from a source of the same
    size, fixed for {1,1…}; FixedEnds->{ψs} specifies an
    arbitrary source population.  The fitness is represented
    by a function W[i,j,h] in the symmetrical model, W[X,Y]
    if all 2^n genotypes are iterated, or W[p]={W̄,{s1,…}} if
    only AlleleFrequencies are supplied.  If ψ consists of a
    list of demes, migration is at a rate m/2 to each neighbour.
    The fitness is then represented by a function W[deme][i,j,h]
    in the symmetrical model, or W[deme][X,Y] if all 2^n
    genotypes are iterated. Compiled->True uses compiled code.
    The default order is random union→migration→selection→meiosis,
    which is represented by MigrationOrder→UMS. MigrationOrder→MUS
    represents migration→random union→selection→meiosis.
    MigrationOrder→USM represents random
    union→selection→migration→meiosis.  For a cline,
    FixedEnds→{ψ0,ψ1} fixes genotype frequencies at ψ0, ψ1
    at either end. For SymmetricNeutralHaploidFrequencies,
    IterateExact[{ψ,u},W,{m,u*}] includes the frequency of
    the neutral marker after one generation of selection
    followed by immigration from a source population with
    frequency u*.  DemeSize->F allows deme size to change to
    some function F[n,W̄], where n is the number before
    selection.  Compiled code can only use a limited set of
    F: either n*&, where n* is a fixed number; or ExponentialF[r,β,λ],
    which represents rnW̄^β Exp[-λn]; or PowerF[r,β,γ], which
    represents rSuperscriptBox[n,1 - γ]W̄^β. RandomDrift->True
    includes random sampling of a number of haploid gametes
    equal to the deme size after selection.";

Options[IterateExact] = {Compiled → False, MigrationOrder → UMS,
   FixedEnds → {{}, {}}, DemeSize → (#1 &), RandomDrift → False};

IterateExact::badMigrationOrder =
  "`1` is not a valid setting for MigrationOrder";
```

IterateExact – one deme

```
IterateExact[(pr : NonNeutralRepresentations)[ψ_, nd_], ww_, m_,
   opts___Rule] := Module[{i, gg, ψs, f, cr, gt, dt, ht,
    iQ = MatchQ[pr, HaploidIndividuals],
    cQ = Compiled /. {opts} /. Options[IterateExact],
    mo = MigrationOrder /. {opts} /. Options[IterateExact],
    fe = FixedEnds /. {opts} /. Options[IterateExact],
    rd = RandomDrift /. {opts} /. Options[IterateExact],
    ng = NumberOfGenes[pr[ψ, nd]], spr},
   ψs = If[fe[[1]] === {} ⋁ fe[[1]] === {{}},
     spr = Switch[pr,
       HaploidIndividuals, HaploidFrequencies,
       _, pr];
     MakePopulation[ng, 1, nd, NumericalModel → spr],
     fe[[1]]];
   If[cQ,
    f = DemeSize /. {opts} /. Options[IterateExact];
    f = f /. {((nn_?NumberQ) &) → {nn, 0, 1, 0}, (#1 &) → {1, 0, 0, 0},
       PowerF[r_, β_, γ_] → {r, β, γ, 0},
       ExponentialF[r_, β_, λ_] → {r, β, 0, λ},
       (PowerF | ExponentialF)[r_, β_, γ_, λ_] → {r, β, γ, λ},
       ExponentialF[r_, β_, λ_] → {r, β, 0, λ}};
    cr = If[pr === AlleleFrequencies,
      IterateCompiled[ww, oneDeme, mo][ψ, nd, m, ψs[[1]], ψs[[2]], f, rd],
      gt = GameteTable[ng, ww, NumericalModel → pr, opts];
      If[iQ,
       ht = HaploidTypes[Log[2, Length[ψs[[1]]]], opts];
       Flatten[Outer[List, ht, ht, 1], 1];
       IterateCompiled[oneDeme, ind, mo][ψ, nd, gt, dt, m, ψs[[1]], ψs[[2]], f],
       IterateCompiled[oneDeme, mo][ψ, nd, gt, m, ψs[[1]], ψs[[2]], f, rd]]];
    pr[Drop[cr, -1], Last[cr]],
    Switch[mo,
     UMS,
     NewGametes[MigrateExact[MakeDiploid[pr[ψ, nd]], MakeDiploid[ψs], m],
      ww, opts],
     MUS,
     NewGametes[MakeDiploid[MigrateExact[pr[ψ, nd], ψs, m]], ww, opts],
     USM,
     ψ2 = If[iQ ⋀ OddQ[nd], MakeDiploid[pr[Drop[ψ, 1], nd - 1]],
       MakeDiploid[pr[ψ, nd]]];
     MigrateExact[NewGametes[ψ2, ww, opts], ψs, m],
     _, Message[IterateExact::badMigrationOrder, mo]; pr[ψ, nd]]]];
```

IterateExact – cline

```
IterateExact[ψ : {NonNeutralRepresentations[_, _] ..}, ww_, m_, opts___] :=
  Module[{i, gg, nd = Length[ψ], cr, gt, ψ0, ψ1, n0, n1, pr = Head[ψ[[1]]],
    cQ = Compiled /. {opts} /. Options[IterateExact],
    mo = MigrationOrder /. {opts} /. Options[IterateExact],
    fe = FixedEnds /. {opts} /. Options[IterateExact],
    rd = RandomDrift /. {opts} /. Options[IterateExact],
    ϕ, nl, dsrl,
    ng = NumberOfGenes[ψ]},
    If[cQ,
    ϕ = First /@ ψ; nl = Last /@ ψ;
    f = DemeSize /. {opts} /. Options[IterateExact];
    f = f /. {((nn_?NumberQ) &) → Array[{nn, 0, 1, 0} &, nd],
       (#1 &) → Array[{1, 0, 0, 0} &, nd],
       PowerF[r_, β_, γ_] → Array[{r, β, γ, 0} &, nd],
       ExponentialF[r_, β_, λ_] → Array[{r, β, 0, λ} &, nd],
       (PowerF | ExponentialF)[r_, β_, γ_, λ_] → Array[{r, β, γ, λ} &, nd],
       ExponentialF[r_, β_, λ_] → Array[{r, β, 0, λ} &, nd]};
    If[fe[[1]] === {}, ψ0 = {}; n0 = 0, ψ0 = fe[[1, 1]]; n0 = fe[[1, 2]]];
    If[fe[[-1]] === {}, ψ1 = {}; n1 = 0, ψ1 = fe[[-1, 1]]; n1 = fe[[-1, 2]]];
    cr = If[pr === AlleleFrequencies,
      IterateCompiled[ww, cline, mo][ϕ, nl, m, ψ0, ψ1, n0, n1, f, rd],
      gt = Table[GameteTable[ng, ww[i], NumericalModel → Head[ψ[[1]]], opts],
        {i, nd}];
      IterateCompiled[cline, mo][ϕ, nl, gt, m, ψ0, ψ1, n0, n1, f, rd]];
    pr[Drop[#, -1], Last[#]] & /@ cr,
    dsrl[j_] := {(DemeSize → f_?(Not[ListQ[#]] &)) :> (DemeSize → f[j]),
      (DemeSize → f_List) :> (DemeSize → f[[j]])};
    Switch[mo,
     UMS,
     gg = MigrateExact[MakeDiploid[ψ], m, FixedEnds → (MakeDiploid /@ fe)];
     Table[NewGametes[gg[[i]], ww[i], ##] &@@ ({opts} /. dsrl[i]), {i, nd}],
     MUS,
     gg = MakeDiploid[MigrateExact[ψ, m, FixedEnds → fe]];
     Table[NewGametes[gg[[i]], ww[i], ##] &@@ ({opts} /. dsrl[i]), {i, nd}],
     USM,
      gg = MakeDiploid[ψ];
     MigrateExact[Table[NewGametes[gg[[i]], ww[i], ##] &@@
        ({opts} /. dsrl[i]), {i, nd}], m, FixedEnds → fe],
     _,
     Message[IterateExact::badMigrationOrder, mo]; pr[ψ, n]]]];
```

IterateExact – one deme, compiled

```
IterateCompiled[oneDeme, UMS] =
  Compile[{{ψ, _Real, 1}, n, {gt, _Real, 3}, m, {ψs, _Real, 1}, ns,
    {F, _Real, 1}, {rd, True | False}},
    Module[{tt, wb, nm, rnm},
    nm = m ns + (1 - m) n;
    tt =
     (Plus @@
        Plus @@
         (gt * (n (1 - m) Outer[Times, ψ, ψ] + ns m Outer[Times, ψs, ψs]))) / nm;
    wb = Plus @@ tt;
    nm = F[[1]] nm^(1-F[[3]]) wb^F[[2]] Exp[- nm F[[4]]];
    Append[If[rd,
      rnm = Max[1, Round[nm]];
      RandomMultinomial[rnm, tt/wb] / rnm,
      tt/wb], nm]],
      {{wb, _Real}, {nm, _Real}, {rnm, _Integer}, {tt, _Real, 1},
    {RandomMultinomial[_, _], _Integer, 1}}];

IterateCompiled[oneDeme, MUS] =
  Compile[{{ψ, _Real, 1}, n, {gt, _Real, 3}, m, {ψs, _Real, 1}, ns,
    {F, _Real, 1}, {rd, True | False}},
   Module[{tt, wb, nm, rnm},
    nm = m ns + (1 - m) n;
    tt = (n (1 - m) ψ + ns m ψs) / nm;
    tt = Plus @@ Plus @@ (gt * Outer[Times, tt, tt]);
    wb = Plus @@ tt;
    nm = F[[1]] nm^(1-F[[3]]) wb^F[[2]] Exp[- nm F[[4]]];
    Append[If[rd,
      rnm = Max[1, Round[nm]];
      RandomMultinomial[rnm, tt/wb] / rnm,
      tt/wb], nm]],
      {{wb, _Real}, {nm, _Real}, {rnm, _Integer}, {tt, _Real, 1},
    {RandomMultinomial[_, _], _Integer, 1}}];
```

```
IterateCompiled[oneDeme, USM] =
  Compile[{{ψ, _Real, 1}, n, {gt, _Real, 3}, m, {ψs, _Real, 1}, ns,
    {F, _Real, 1}, {rd, True | False}},
   Module[{tt, wb, nn, nnn, rnm},
    tt = Plus @@ Plus @@ (gt * Outer[Times, ψ, ψ]);
    wb = Plus @@ tt; nn = F[[1]] n^(1-F[[3]]) wb^F[[2]] Exp[- n F[[4]]];
    nnn = (1 - m) nn + ns m;
    Append[If[rd,
      rnm = Max[1, Round[nnn]];
      RandomMultinomial[rnm, (nn (1 - m) tt/wb + ns m ψs)/nnn] / rnm,
      (nn (1 - m) tt/wb + ns m ψs)/nnn], nnn]],
   {{wb, _Real}, {nn, _Real}, {rnm, _Integer}, {nnn, _Real},
    {tt, _Real, 1}, {RandomMultinomial[_, _], _Integer, 1}}];
```

IterateExact – one deme, compiled, allele frequencies

```
IterateCompiled[ww_, oneDeme, UMS] :=
  Module[{fw},
   fw[p_] := Flatten[ww[p]];
   IterateCompiled[ww, oneDeme, UMS] =
    Compile[{{p, _Real, 1}, n, m, {ps, _Real, 1}, ns, {F, _Real, 1},
      {rd, True | False}},
     Module[{wv, np, wb, nm, rnm},
      nm = m ns + (1 - m) n;
      np = (n (1 - m) p + ns m ps)/nm;
      wv = fw[np]; wb = wv[[1]];
      np = np (1 + (1 - np) Drop[wv, 1]/wb);
      nm = F[[1]] nm^(1-F[[3]]) wb^F[[2]] Exp[- nm F[[4]]];
      Append[If[rd,
        rnm = Max[1, Round[nm]];
        (First[RandomMultinomial[rnm, {#, 1 - #}]] & /@ np) / rnm,
        np],
       nm]],
     {{wv, _Real, 1}, {np, _Real, 1}, {wb, _Real}, {nm, _Real},
      {rnm, _Integer}, {RandomMultinomial[_, _], _Integer, 1},
      {fw[_], _Real, 1}}]];

IterateCompiled[ww_, oneDeme, MUS] := IterateCompiled[ww, oneDeme, UMS];
```

```
IterateCompiled[ww_, oneDeme, USM] :=
  Module[{fw},
   fw[p_] := Flatten[ww[p]];
   IterateCompiled[ww, oneDeme, USM] =
    Compile[{{p, _Real, 1}, n, m, {ps, _Real, 1}, ns, {F, _Real, 1},
      {rd, True | False}},
     Module[{np, wb, nn, nnn, rnm, wv},
      wv = fw[p]; wb = wv[[1]];
      np = p (1 + (1 - p) Drop[wv, 1]/wb);
      nn = F[[1]] n^(1-F[[3]]) wb^F[[2]] Exp[- n F[[4]]];
      nnn = (1 - m) nn + ns m;
      Append[If[rd,
        rnm = Max[1, Round[nnn]];
        (First[RandomMultinomial[rnm, {#, 1 - #}]] & /@
           (nn (1 - m) np + ns m ps)/nnn)/rnm,
        (nn (1 - m) np + ns m ps)/nnn], nnn]],
     {{wb, _Real}, {wv, _Real, 1}, {nn, _Real}, {rnm, _Integer},
      {nnn, _Real}, {np, _Real, 1},
      {RandomMultinomial[_, _], _Integer, 1}}]];
```

IterateExact – cline, compiled

```
IterateCompiled[cline, UMS] =
  Compile[{{ψ, _Real, 2}, {nl, _Real, 1}, {gt, _Real, 4}, m,
    {fe0, _Real, 1}, {fe1, _Real, 1}, n0, n1, {F, _Real, 2},
    {rd, True | False}}, Module[{ϕ, nϕ, nn, tt, i, nm, wb, rnm, nnm},
    ϕ = Outer[Times, #, #] & /@ ψ;
    ϕ = Prepend[Append[ϕ,
       If[Length[fe1] == 0, Last[ϕ], Outer[Times, fe1, fe1]]],
      If[Length[fe0] == 0, First[ϕ], Outer[Times, fe0, fe0]]];
    nn = Prepend[Append[nl,
       If[Length[fe1] == 0, Last[nl], n1]],
      If[Length[fe0] == 0, First[nl], n0]]; nϕ = nn * ϕ;
    nϕ = Drop[Drop[(1 - m) nϕ + m/2 (RotateLeft[nϕ] + RotateRight[nϕ]), 1],
      -1];
    nm = Drop[Drop[(1 - m) nn + m/2 (RotateLeft[nn] + RotateRight[nn]), 1], -1];
    Table[tt = (Plus @@ Plus @@ (gt[[i]] * nϕ[[i]])) / nm[[i]]; wb = Plus @@ tt;
     nnm = F[[i, 1]] nm[[i]]^(1-F[[i,3]]) wb^F[[i,2]] Exp[- nm[[i]] F[[i, 4]]];
     rnm = If[rd, RandomReal[PoissonDistribution[nnm]],
       Max[1, Round[nnm]]];
     Append[If[rd, RandomMultinomial[rnm, tt/wb] / rnm, tt/wb], nnm],
     {i, Length[ψ]}]],
      {{tt, _Real, 1}, {nm, _Real, 1}, {wb, _Real}, {nn, _Real, 1},
    {ϕ, _Real, 3}, {nϕ, _Real, 3}, {i, _Integer},
    {RandomMultinomial[_, _], _Integer, 1},
    {RandomReal[PoissonDistribution[_]], _Integer}, {rnm, _Integer},
    {nnm, _Real}}];
```

```
IterateCompiled[cline, MUS] =
  Compile[{{ψ, _Real, 2}, {nl, _Real, 1}, {gt, _Real, 4}, m,
    {fe0, _Real, 1}, {fe1, _Real, 1}, n0, n1, {F, _Real, 2},
    {rd, True | False}}, Module[{ϕ, y, ny, nn, nm, nnm, rnm, tt, wb},
    (y = Prepend[Append[ψ,
        If[Length[fe1] == 0, Last[ψ], fe1]],
       If[Length[fe0] == 0, First[ψ], fe0]];
     nn = Prepend[Append[nl,
        If[Length[fe1] == 0, Last[nl], n1]],
       If[Length[fe0] == 0, First[nl], n0]];
     ny = nn * y;
     ny = Drop[Drop[(1 - m) ny + m/2 (RotateLeft[ny] + RotateRight[ny]), 1],
       -1];
     nm = Drop[Drop[(1 - m) nn + m/2 (RotateLeft[nn] + RotateRight[nn]), 1],
       -1]; ϕ = Outer[Times, #, #] & /@ (ny/nm);
     Table[tt = Plus @@ Plus @@ (gt[[i]] * ϕ[[i]]); wb = Plus @@ tt;
      nnm = F[[i, 1]] wb^F[[i,2]] nm[[i]]^(1-F[[i,3]]) Exp[- nm[[i]] F[[i, 4]]];
      rnm = If[rd, RandomReal[PoissonDistribution[nnm]],
        Max[1, Round[nnm]]];
      Append[If[rd, RandomMultinomial[rnm, tt/wb]/rnm, tt/wb], nnm],
      {i, Length[ψ]}])], {{nn, _Real, 1}, {nm, _Real, 1}, {wb, _Real},
    {tt, _Real, 1}, {y, _Real, 2}, {ny, _Real, 2}, {ϕ, _Real, 3},
    {i, _Integer}, {RandomMultinomial[_, _], _Integer, 1},
    {RandomReal[PoissonDistribution[_]], _Integer}, {rnm, _Integer},
    {nnm, _Real}}];
```

```
IterateCompiled[cline, USM] =
  Compile[{{ψ, _Real, 2}, {nl, _Real, 1}, {gt, _Real, 4}, m,
    {fe0, _Real, 1}, {fe1, _Real, 1}, n0, n1, {F, _Real, 2},
    {rd, True | False}},
   Module[{ϕ, nϕ, nn, nm, rnm, wb, ny, y, y0, y1, tt, nd = Length[ψ], newy},
    (ϕ = Outer[Times, #, #] & /@ ψ;
     y = Table[Plus @@ Plus @@ (gt[[i]] * ϕ[[i]]), {i, nd}];
     nn = Table[wb = Plus @@ y[[i]]; F[[i, 1]] nl[[i]]^(1-F[[i,3]]) wb^F[[i,2]]
        Exp[- nl[[i]] F[[i, 4]]], {i, nd}];
     y = Table[If[rd,
        rnm = RandomReal[PoissonDistribution[nn[[i]]]];
        RandomMultinomial[rnm, y[[i]]/Plus @@ y[[i]]]/rnm,
        y[[i]]/Plus @@ y[[i]]], {i, nd}];
     newy = Prepend[Append[y, If[Length[fe1] == 0, Last[y], fe1]],
       If[Length[fe0] == 0, First[y], fe0]];
     nm = Prepend[Append[nn, If[Length[fe1] == 0, Last[nn], n1]],
       If[Length[fe0] == 0, First[nn], n0]];
     ny = newy * nm;
     nm = Drop[Drop[(1 - m) nm + m/2 (RotateLeft[nm] + RotateRight[nm]), 1],
       -1];
     y = Drop[Drop[(1 - m) ny + m/2 (RotateLeft[ny] + RotateRight[ny]), 1], -1]/
       nm;
     Table[Append[y[[i]], nm[[i]]], {i, nd}])],
   {{nn, _Real, 1}, {nm, _Real, 1}, {rnm, _Integer}, {wb, _Real},
    {y, _Real, 2}, {newy, _Real, 2}, {ny, _Real, 2}, {tt, _Real, 1},
    {ϕ, _Real, 3}, {nϕ, _Real, 3}, {i, _Integer},
    {RandomMultinomial[_, _], _Integer, 1},
    {RandomReal[PoissonDistribution[_]], _Integer}, {rnm, _Integer},
    {nnm, _Real}}];
```

IterateExact – cline, compiled, allele frequencies

```
IterateCompiled[ww_, cline, UMS] := Module[{fw},
  fw[p_, d_] := Flatten[ww[d][p]];
  IterateCompiled[ww, cline, UMS] =
   Compile[{{ψ, _Real, 2}, {nl, _Real, 1}, m, {fe0, _Real, 1},
     {fe1, _Real, 1}, n0, n1, {F, _Real, 2}, {rd, True | False}},
    Module[{nϕ, nn, np, tt, i, nm, wb, rnm, nnm, wv, ψn},
     ψn = Prepend[Append[ψ,
        If[Length[fe1] == 0, Last[ψ], fe1]],
       If[Length[fe0] == 0, First[ψ], fe0]]; nn = Prepend[Append[nl,
        If[Length[fe1] == 0, Last[nl], n1]],
       If[Length[fe0] == 0, First[nl], n0]]; nϕ = nn * ψn;
     nϕ = Drop[Drop[(1 - m) nϕ + m/2 (RotateLeft[nϕ] + RotateRight[nϕ]), 1],
       -1];
     nm = Drop[Drop[(1 - m) nn + m/2 (RotateLeft[nn] + RotateRight[nn]), 1],
       -1]; Table[np = nϕ[[i]]/nm[[i]]; wv = fw[np, i]; wb = wv[[1]];
      np = np (1 + (1 - np) Drop[wv, 1]/wb);
      nnm = F[[i, 1]] nm[[i]]^(1-F[[i,3]]) wb^F[[i,2]] Exp[- nm[[i]] F[[i, 4]]];
      Append[If[rd,
        rnm = Max[1, Round[nnm]];
        (First[RandomMultinomial[rnm, {#, 1 - #}]] & /@ np) / rnm,
        np], nnm], {i, Length[ψ]}]],
     {{ψn, _Real, 2}, {wv, _Real, 1}, {nm, _Real, 1},
     {wb, _Real}, {nn, _Real, 1}, {np, _Real}, {nϕ, _Real, 3},
     {i, _Integer}, {RandomMultinomial[_, _], _Integer, 1},
     {rnm, _Integer}, {nnm, _Real}, {fw[_, _], _Real, 1}}]];

IterateCompiled[ww_, cline, MUS] := IterateCompiled[ww, cline, UMS];
```

```
IterateCompiled[ww_, cline, USM] := Module[{fw},
  fw[p_, d_] := Flatten[ww[d][p]];
  IterateCompiled[ww, cline, USM] =
   Compile[{{ϕ, _Real, 2}, {nl, _Real, 1}, m, {fe0, _Real, 1},
     {fe1, _Real, 1}, n0, n1, {F, _Real, 2}, {rd, True | False}},
    Module[{nϕ, nn, nm, rnm, ny, y, y0, y1, tt, nd = Length[ϕ], newy, wvl},
     (wvl = Table[fw[ϕ[[i]], i], {i, nd}];
      y = Table[ϕ[[i]] (1 + (1 - ϕ[[i]]) Drop[wvl[[i]], 1]/wvl[[i, 1]]), {i, nd}];
      nn = Table[F[[i, 1]] nl[[i]]^(1-F[[i,3]]) wvl[[i, 1]]^F[[i,2]] Exp[- nl[[i]] F[[i, 4]]],
        {i, nd}];
      y = Table[If[rd,
         rnm = Max[1, Round[nn[[i]]]];
         (First[RandomMultinomial[rnm, {#, 1 - #}]] & /@ y[[i]])/rnm,
         y[[i]]], {i, nd}];
      newy = Prepend[Append[y, If[Length[fe1] == 0, Last[y], fe1]],
        If[Length[fe0] == 0, First[y], fe0]];
      nm = Prepend[Append[nn, If[Length[fe1] == 0, Last[nn], n1]],
        If[Length[fe0] == 0, First[nn], n0]];
      ny = newy * nm;
      nm = Drop[Drop[(1 - m) nm + m/2 (RotateLeft[nm] + RotateRight[nm]), 1],
        -1];
      y = Drop[Drop[(1 - m) ny + m/2 (RotateLeft[ny] + RotateRight[ny]), 1],
         -1]/nm;
      Table[Append[y[[i]], nm[[i]]], {i, nd}])],
    {{nn, _Real, 1}, {nm, _Real, 1}, {rnm, _Integer}, {wvl, _Real, 2},
     {y, _Real, 2}, {newy, _Real, 2}, {ny, _Real, 2}, {tt, _Real, 1},
     {nϕ, _Real, 3}, {i, _Integer}, {RandomMultinomial[_, _], _Integer, 1},
     {rnm, _Integer}, {nnm, _Real}, {fw[_, _], _Real, 1}}]];
```

IterateExact – one deme, symmetric neutral

```
IterateExact[ϕ : SymmetricNeutralHaploidFrequencies[{ψ_, u_}, nd_],
   ww_, m_, opts___Rule] := Module[{i, gg, ψs, f, cr, gt,
    cQ = Compiled /. {opts} /. Options[IterateExact],
    mo = MigrationOrder /. {opts} /. Options[IterateExact],
    fe = FixedEnds /. {opts} /. Options[IterateExact],
    ng = NumberOfGenes[ϕ]},
   ψs = If[fe[[1]] === {} ⋁ fe[[1]] === {{}},
     MakePopulation[ng, {1, 1}, nd,
      NumericalModel → SymmetricNeutralHaploidFrequencies],
     fe[[1]]];
   If[cQ,
    f = DemeSize /. {opts} /. Options[IterateExact];
    f = f /. {((nn_?NumberQ) &) → {nn, 0, 1, 0}, (#1 &) → {1, 0, 0, 0},
       PowerF[r_, β_, γ_] → {r, β, γ, 0},
       ExponentialF[r_, β_, λ_] → {r, β, 0, λ},
       (PowerF | ExponentialF)[r_, β_, γ_, λ_] → {r, β, γ, λ},
       ExponentialF[r_, β_, λ_] → {r, β, 0, λ}};
    gt = GameteTable[ng, ww, NumericalModel → SymmetricHaploidFrequencies,
      opts];
    cr = IterateCompiled[neutral, oneDeme, mo][ψ, u, nd, gt, m,
      ψs[[1, 1]], ψs[[1, 2]], ψs[[2]], f];
    SymmetricNeutralHaploidFrequencies[Partition[Drop[cr, -1], ng + 1],
     Last[cr]],
    Switch[mo,
     UMS, NewGametes[MigrateExact[MakeDiploid[ϕ], MakeDiploid[ψs], m],
      ww, opts],
     MUS, NewGametes[MakeDiploid[MigrateExact[ϕ, ψs, m]], ww, opts],
     USM, MigrateExact[NewGametes[MakeDiploid[ϕ], ww, opts], ψs, m],
     _, Message[IterateExact::badMigrationOrder, mo]; ϕ]]];
```

IterateExact – one deme, compiled, symmetric neutral

```
IterateCompiled[neutral, oneDeme, UMS] =
  Compile[{{ψ, _Real, 1}, {u, _Real, 1}, n, {gt, _Real, 3}, m,
    {ψs, _Real, 1}, {us, _Real, 1}, ns, {F, _Real, 1}},
    Module[{tt, ttu, wb, nm},
    nm = m ns + (1 - m) n;
    tt =
     (Plus @@
        Plus @@
         (gt * (n (1 - m) Outer[Times, ψ, ψ] + ns m Outer[Times, ψs, ψs]))) / nm;

    ttu =
     (Plus @@
        Plus @@
         (gt * (n (1 - m) Outer[Times, ψ, ψ] Outer[Plus, u, u] +
             ns m Outer[Times, ψs, ψs] Outer[Plus, us, us]))) / (2 nm);
    wb = Plus @@ tt;
    nm = F[[1]] nm^(1-F[[3]]) wb^F[[2]] Exp[- nm F[[4]]];
    Append[Join[tt/wb, DivideZero[ttu, tt]], nm]],
        {{wb, _Real}, {nm, _Real}, {tt, _Real, 1}, {ttu, _Real, 1},
    {DivideZero[_, _], _Real, 1}}];

IterateCompiled[neutral, oneDeme, MUS] =
  Compile[{{ψ, _Real, 1}, {u, _Real, 1}, n, {gt, _Real, 3}, m,
    {ψs, _Real, 1}, {us, _Real, 1}, ns, {F, _Real, 1}},
   Module[{tt, ttu, wb, nm, nu, ngt},
    nm = m ns + (1 - m) n;
    tt = (n (1 - m) ψ + ns m ψs) / nm;
    ttu = (n (1 - m) ψ u + ns m ψs us) / nm;
    nu = DivideZero[ttu, tt];
    ngt = gt * Outer[Times, tt, tt];
    tt = Plus @@ Plus @@ ngt;
    ttu = (Plus @@ Plus @@ (ngt * Outer[Plus, nu, nu])) / 2;
    wb = Plus @@ tt;
    nm = F[[1]] nm^(1-F[[3]]) wb^F[[2]] Exp[- nm F[[4]]];
    Append[Join[tt/wb, DivideZero[ttu, tt]], nm]],
        {{wb, _Real}, {nm, _Real}, {tt, _Real, 1}, {ttu, _Real, 1},
    {ngt, _Real, 3}, {nu, _Real, 1}, {DivideZero[_, _], _Real, 1}}];
```

```
IterateCompiled[neutral, oneDeme, USM] =
  Compile[{{ψ, _Real, 1}, {u, _Real, 1}, n, {gt, _Real, 3}, m,
    {ψs, _Real, 1}, {us, _Real, 1}, ns, {F, _Real, 1}},
   Module[{tt, ttu, wb, nn, nnn, ttn},
    tt = Plus @@ Plus @@ (gt * Outer[Times, ψ, ψ]);
    ttu = (Plus @@ Plus @@ (gt * Outer[Times, ψ, ψ] * Outer[Plus, u, u])) / 2;
    wb = Plus @@ tt;
    nn = F[[1]] n^(1-F[[3]]) wb^F[[2]] Exp[- n F[[4]]];
    nnn = (1 - m) nn + ns m;
    ttn = nn (1 - m) tt/wb + ns m ψs;
    Append[Join[ttn/nnn, DivideZero[nn (1 - m) ttu/wb + ns m ψs us, ttn]], nnn],
   {{wb, _Real}, {nn, _Real}, {nnn, _Real}, {tt, _Real, 1},
    {ttn, _Real, 1}, {ttu, _Real, 1}, {DivideZero[_, _], _Real, 1}}];
```

IterateExact – cline, symmetric neutral

```
IterateExact[ψ : {SymmetricNeutralHaploidFrequencies[{_, _}, _] ..},
   ww_, m_, opts___] :=
  Module[{i, gg, nd = Length[ψ], cr, gt, ψ0, ψ1, u0, u1, n0, n1,
    cQ = Compiled /. {opts} /. Options[IterateExact],
    mo = MigrationOrder /. {opts} /. Options[IterateExact],
    fe = FixedEnds /. {opts} /. Options[IterateExact],
    ϕ, u, nl, dsrl,
    ng = NumberOfGenes[ψ]},
    If[cQ,
    ϕ = First /@ First /@ ψ;
    u = Last /@ First /@ ψ;
    nl = Last /@ ψ;
    f = DemeSize /. {opts} /. Options[IterateExact];
    f = f /. {((nn_?NumberQ) &) → Array[{nn, 0, 1, 0} &, nd],
       (#1 &) → Array[{1, 0, 0, 0} &, nd],
       PowerF[r_, β_, γ_] → Array[{r, β, γ, 0} &, nd],
       ExponentialF[r_, β_, λ_] → Array[{r, β, 0, λ} &, nd],
       (PowerF | ExponentialF)[r_, β_, γ_, λ_] → Array[{r, β, γ, λ} &, nd],
       ExponentialF[r_, β_, λ_] → Array[{r, β, 0, λ} &, nd]};
    If[fe[[1]] === {},
     ψ0 = {}; u0 = {}; n0 = 0,
     ψ0 = fe[[1, 1, 1]]; u0 = fe[[1, 1, 2]]; n0 = fe[[1, 2]]];
    If[fe[[-1]] === {},
     ψ1 = {}; u1 = {}; n1 = 0,
     ψ1 = fe[[-1, 1, 1]]; u1 = fe[[-1, 1, 2]]; n1 = fe[[-1, 2]]];
    gt = Table[GameteTable[ng, ww[i],
       NumericalModel → SymmetricNeutralHaploidFrequencies, opts],
      {i, nd}];
    cr = IterateCompiled[neutral, cline, mo][ϕ, u, nl, gt, m, ψ0,
      ψ1, u0, u1, n0, n1, f];
    SymmetricNeutralHaploidFrequencies[Partition[Drop[#, -1], ng + 1],
       Last[#]] & /@ cr,
    dsrl[j_] := {(DemeSize → f_?(Not[ListQ[#]] &)) :> (DemeSize → f[j]),
      (DemeSize → f_List) :> (DemeSize → f[[j]])};
    Switch[mo,
     UMS,
     gg = MigrateExact[MakeDiploid[ψ], m, FixedEnds → (MakeDiploid /@ fe)];
     Table[NewGametes[gg[[i]], ww[i], ##] & @@ ({opts} /. dsrl[i]), {i, nd}],
     MUS,
     gg = MakeDiploid[MigrateExact[ψ, m, FixedEnds → fe]];
     Table[NewGametes[gg[[i]], ww[i], ##] & @@ ({opts} /. dsrl[i]), {i, nd}],
     USM,
      gg = MakeDiploid[ψ];
     MigrateExact[Table[NewGametes[gg[[i]], ww[i], ##] & @@
        ({opts} /. dsrl[i]), {i, nd}], m, FixedEnds → fe],
     _,
     Message[IterateExact::badMigrationOrder, mo];
     SymmetricNeutralHaploidFrequencies[ψ, n]]]];
```

IterateExact – cline, compiled, symmetric neutral

```
IterateCompiled[neutral, cline, UMS] =
  Compile[{{ψ, _Real, 2}, {u, _Real, 2}, {nl, _Real, 1}, {gt, _Real, 4},
    m, {fe0, _Real, 1}, {fe1, _Real, 1}, {feU0, _Real, 1},
    {feU1, _Real, 1}, n0, n1, {F, _Real, 2}},
   Module[{ϕ, ϕu, nϕ, nϕu, nn, tt, ttu, nu, i, nm, wb, nnm},
    ϕ = Outer[Times, #, #] & /@ ψ;
    ϕu = MapThread[Outer[Times, #1, #1] Outer[Plus, #2, #2] &, {ψ, u}] / 2;
    ϕ = Prepend[Append[ϕ,
       If[Length[fe1] == 0, Last[ϕ], Outer[Times, fe1, fe1]]],
      If[Length[fe0] == 0, First[ϕ], Outer[Times, fe0, fe0]]];
    ϕu = Prepend[Append[ϕu,
       If[Length[fe1] == 0, Last[ϕu],
        Outer[Times, fe1, fe1] Outer[Plus, feU1, feU1] / 2]],
      If[Length[fe0] == 0, First[ϕu],
       Outer[Times, fe0, fe0] Outer[Plus, feU0, feU0] / 2]];
    nn = Prepend[Append[nl,
       If[Length[fe1] == 0, Last[nl], n1]],
      If[Length[fe0] == 0, First[nl], n0]];
    nϕ = nn * ϕ; nϕu = nn * ϕu;
    nϕ = Drop[Drop[(1 - m) nϕ + m/2 (RotateLeft[nϕ] + RotateRight[nϕ]), 1],
      -1];
    nϕu = Drop[Drop[(1 - m) nϕu + m/2 (RotateLeft[nϕu] + RotateRight[nϕu]), 1],
      -1];
    nm = Drop[Drop[(1 - m) nn + m/2 (RotateLeft[nn] + RotateRight[nn]), 1], -1];
    nu = MapThread[DivideZero[#1, #2] &, {nϕu, nϕ}, 2];
    Table[
     tt = (Plus @@ Plus @@ (gt[[i]] * nϕ[[i]])) / nm[[i]];
     ttu = (Plus @@ Plus @@ (gt[[i]] * nϕ[[i]] * nu[[i]])) / nm[[i]];
     wb = Plus @@ tt;
     nnm = F[[i, 1]] nm[[i]]^(1-F[[i,3]]) wb^F[[i,2]] Exp[- nm[[i]] F[[i, 4]]];
     Append[Join[tt/wb, DivideZero[ttu, tt]], nnm],
     {i, Length[ψ]}]],
      {{tt, _Real, 1}, {ttu, _Real, 1}, {nm, _Real, 1},
    {wb, _Real}, {nn, _Real, 1}, {ϕ, _Real, 3}, {ϕu, _Real, 3},
    {nu, _Real, 3}, {nϕ, _Real, 3}, {nϕu, _Real, 3}, {i, _Integer},
    {nnm, _Real}, {DivideZero[_, _], _Real, 1}}];
```

```
IterateCompiled[neutral, cline, MUS] =
  Compile[{{ψ, _Real, 2}, {u, _Real, 2}, {nl, _Real, 1}, {gt, _Real, 4},
    m, {fe0, _Real, 1}, {fe1, _Real, 1}, {feU0, _Real, 1},
    {feU1, _Real, 1}, n0, n1, {F, _Real, 2}},
   Module[{ϕ, ϕu, y, yu = ψ * u, ny, nyu, nn, nm, nnm, tt, ttu, wb},
    (y = Prepend[Append[ψ,
        If[Length[fe1] == 0, Last[ψ], fe1]],
       If[Length[fe0] == 0, First[ψ], fe0]];
     yu = Prepend[Append[yu,
        If[Length[fe1] == 0, Last[yu], fe1 * feU1]],
       If[Length[fe0] == 0, First[yu], fe0 * feU0]];
     nn = Prepend[Append[nl,
        If[Length[fe1] == 0, Last[nl], n1]],
       If[Length[fe0] == 0, First[nl], n0]];
     ny = nn * y; nyu = nn * yu;
     ny = Drop[Drop[(1 - m) ny + m/2 (RotateLeft[ny] + RotateRight[ny]), 1],
       -1];
     nyu = Drop[Drop[(1 - m) nyu + m/2 (RotateLeft[nyu] + RotateRight[nyu]), 1],
       -1];
     nm = Drop[Drop[(1 - m) nn + m/2 (RotateLeft[nn] + RotateRight[nn]), 1],
       -1]; ny = ny / nm;
     nyu = MapThread[DivideZero[#1, #2] &, {nyu, ny}];
     ϕ = Outer[Times, #, #] & /@ ny;
     ϕu = MapThread[Outer[Times, #1, #1] Outer[Plus, #2, #2]/2 &, {ny, nyu}];
     Table[
      tt = Plus @@ Plus @@ (gt⟦i⟧ * ϕ⟦i⟧);
      ttu = Plus @@ Plus @@ (gt⟦i⟧ * ϕu⟦i⟧);
      wb = Plus @@ tt;
      nnm = F⟦i, 1⟧ wb^F⟦i,2⟧ nm⟦i⟧^(1-F⟦i,3⟧) Exp[- nm⟦i⟧ F⟦i, 4⟧];
      Append[Join[tt/wb, DivideZero[ttu, tt]], nnm], {i, Length[ψ]}])],
   {{nn, _Real, 1}, {nm, _Real, 1}, {wb, _Real}, {tt, _Real, 1},
    {ttu, _Real, 1}, {y, _Real, 2}, {yu, _Real, 2}, {ny, _Real, 2},
    {nyu, _Real, 2}, {ϕ, _Real, 3}, {ϕu, _Real, 3}, {i, _Integer},
    {DivideZero[_, _], _Real, 1}, {nnm, _Real}}];
```

```
IterateCompiled[neutral, cline, USM] =
  Compile[{{ψ, _Real, 2}, {u, _Real, 2}, {nl, _Real, 1}, {gt, _Real, 4},
    m, {fe0, _Real, 1}, {fe1, _Real, 1}, {feU0, _Real, 1},
    {feU1, _Real, 1}, n0, n1, {F, _Real, 2}},
   Module[{ϕ, nϕ, nn, nu, nm, rnm, wb, ny, y, yu, nyu, y0, y1, tt,
     nd = Length[ψ], newy, newyu},
    (ϕ = Outer[Times, #, #] & /@ ψ; nu = Outer[Plus, #, #]/2 & /@ u;
     y = Table[Plus @@ Plus @@ (gt[[i]] * ϕ[[i]]), {i, nd}];
     yu = Table[Plus @@ Plus @@ (gt[[i]] * ϕ[[i]] * nu[[i]]), {i, nd}];
     yu = DivideZero[yu, y];
     nn = Table[
       wb = Plus @@ y[[i]];
       F[[i, 1]] nl[[i]]^(1 - F[[i, 3]]) wb^F[[i, 2]] Exp[-nl[[i]] F[[i, 4]]], {i, nd}];
     y = Table[y[[i]]/Plus @@ y[[i]], {i, nd}];
     newy = Prepend[Append[y,
        If[Length[fe1] == 0, Last[y], fe1]],
       If[Length[fe0] == 0, First[y], fe0]];
     newyu = Prepend[Append[yu,
        If[Length[fe1] == 0, Last[yu], feU1]],
       If[Length[fe0] == 0, First[yu], feU0]];
     nm = Prepend[Append[nn,
        If[Length[fe1] == 0, Last[nn], n1]],
       If[Length[fe0] == 0, First[nn], n0]];
     ny = newy * nm; nyu = newy * newyu * nm;
     nm = Drop[Drop[(1 - m) nm + m/2 (RotateLeft[nm] + RotateRight[nm]), 1],
       -1];
     y = Drop[Drop[(1 - m) ny + m/2 (RotateLeft[ny] + RotateRight[ny]), 1], -1];
     yu = DivideZero[
       Drop[Drop[(1 - m) nyu + m/2 (RotateLeft[nyu] + RotateRight[nyu]), 1],
        -1], y];
     y = y / nm;
     Table[Append[Join[y[[i]], yu[[i]]], nm[[i]]], {i, nd}])],
    {{nn, _Real, 1}, {nm, _Real, 1}, {rnm, _Integer}, {wb, _Real},
     {y, _Real, 2}, {yu, _Real, 2}, {nyu, _Real, 2}, {newy, _Real, 2},
     {newyu, _Real, 2}, {ny, _Real, 2}, {tt, _Real, 1}, {ϕ, _Real, 3},
     {nu, _Real, 3}, {nϕ, _Real, 3}, {i, _Integer},
     {DivideZero[_, _], _Real, 2}, {rnm, _Integer}, {nnm, _Real}}];
```

```
Lligam::usage =
  "Lligam→{SubscriptBox[r, 1, \ 2],SubscriptBox[r,2, 3],…} is
    an option for Gametes[], GameteTable[] and RandomFamily[]
    which specifies the recombination rates in each interval;
    assumes no interference and a linear map.  The default
    Linkage→False gives unlinked loci. ";
```

MakeCline

```
MakeCline::usage =
  "MakeCline[ndemes,nloci,ninds] sets up a stepped cline, with
    ninds individuals per deme.  MakeCline[ndemes,nloci,ninds,dp]
    sets up a linear gradient, with net change dp<1.
    NumericalModel specifies which representation is to be
    used. SymmetricNeutralModel allows for the presence of a
    neutral locus under the symmetric model; this is initially
    either in a step, or a gradient spanning dp.  ";

Options[MakeCline] = {NumericalModel → HaploidFrequencies};

MakeCline[nd_, ng_, ni_, opts___Rule] :=
 Module[{nm = NumericalModel /. {opts} /. Options[MakeCline],
    pp = Array[1/2 (1 + Sign[# - (nd + 1)/2]) &, nd]},
   MakePopulation[ng, #, ni, opts] & /@
    Switch[nm,
     NonNeutralRepresentations, pp,
     NeutralRepresentations, Transpose[{pp, pp}],
     _, Message[NumericalModel::badSetting, nm]; {}]];

MakeCline[nd_, ng_, ni_, dp_, opts___Rule] :=
 Module[{nm = NumericalModel /. {opts} /. Options[MakeCline],
    pp = Range[(1 - dp)/2, (1 + dp)/2, dp/(nd - 1)]},
   MakePopulation[ng, #, ni, opts] & /@
    Switch[nm,
     NonNeutralRepresentations, pp,
     NeutralRepresentations, Transpose[{pp, pp}],
     _, Message[NumericalModel::badSetting, nm]; {}]];
```

MakeDiploid

```
MakeDiploid::usage =
  "MakeDiploid[HaploidFrequencies[…]] gives the matrix of diploid
    genotype frequencies after random mating. Deme size is
    halved. Applies to vectors of class, genotype and allele
    frequencies;  MakeDiploid[HaploidIndividuals[…]] generates
    a population of diploids by sampling with replacement
    from the haploid population; not particularly useful, and
    added for completeness only.";
```

```
MakeDiploid::odd =
  "MakeDiploid[HaploidIndividuals[ψ,n] only applies when n is
    even.  Deme size `1` has been reduced by 1.";

MakeDiploid[ψ : {PopulationRepresentations[_, _] ...}] := MakeDiploid /@ ψ;
MakeDiploid[ψ : DiploidRepresentations[_, _]] := ψ;
MakeDiploid[HaploidFrequencies[ψ_, n_]] :=
  DiploidFrequencies[Outer[Times, ψ, ψ], n / 2];
MakeDiploid[SymmetricHaploidFrequencies[ψ_, n_]] :=
  SymmetricDiploidFrequencies[Outer[Times, ψ, ψ], n / 2];
MakeDiploid[SymmetricNeutralHaploidFrequencies[{ψ_, u_}, n_]] :=
  SymmetricNeutralDiploidFrequencies[{Outer[Times, ψ, ψ], MidParent[u]},
   n / 2];
MakeDiploid[HaploidIndividuals[ψ_, n_?OddQ]] :=
 (Message[MakeDiploid::odd, n];
  DiploidIndividuals[Partition[Drop[ψ, -1], 2], (n - 1) / 2]);
MakeDiploid[HaploidIndividuals[ψ_, n_?EvenQ]] :=
 DiploidIndividuals[Partition[ψ, 2], n / 2];
```

MakeHaploid

```
MakeHaploid::usage =
  "MakeHaploid[DiploidFrequencies[…]] gives the haploid genotype
    frequencies, averaged over maternal and paternal genomes.
    Deme size is doubled.  Applies to vectors of class,
    genotype and allele frequencies.  Deme size is left
    unaltered, except for MakeHaploid[DiploidIndividuals[…]],
    for which deme size is doubled.  This just flattens the
    list: there is no sampling.";

MakeHaploid[ψ : {DiploidRepresentations[_, _] ...}] := MakeHaploid /@ ψ;
MakeHaploid[DiploidFrequencies[ψ_, n_]] :=
  HaploidFrequencies[(Plus @@ ψ + Plus @@ Transpose[ψ]) / 2, 2 n];
MakeHaploid[AlleleFrequencies[p_, n_]] := AlleleFrequencies[p, 2 n];
MakeHaploid[SymmetricDiploidFrequencies[ψ_, n_]] :=
  SymmetricHaploidFrequencies[(Plus @@ ψ + Plus @@ Transpose[ψ]) / 2, 2 n];
MakeHaploid[SymmetricNeutralDiploidFrequencies[{ψ_, u_}, n_]] :=
  Module[{ψm = (Plus @@ ψ + Plus @@ Transpose[ψ]) / 2, ψu = ψ * u},
   SymmetricNeutralHaploidFrequencies[
    {ψm, (Plus @@ ψu + Plus @@ Transpose[ψu]) / (2 ψm)}, 2 n]];
MakeHaploid[DiploidIndividuals[ψ_, n_]] :=
  HaploidIndividuals[Flatten[ψ, 1], 2 n];
```

MakeIndividuals

```
MakeIndividuals::usage =
  "MakeIndividuals[ψ,n] generates n random individuals from the
    population ψ.  If ψ represents a diploid population, the
    representation DiploidIndividuals[] is returned; similarly
    for haploids.  MakeIndividuals[AlleleFrequencies[p,n0],n]
    returns n haploid individuals, drawn from a population
    at linkage equilibrium.
    MakeIndividuals[SymmetricNeutralHaploidFrequencies[{ψ,u},n0],n]
    adds a neutral marker locus at the last position of the
    genotype.  MakeIndividuals[{ψ1…},{n1,…}] generates
    individuals across a cline.  MakeIndividuals[j,n] returns
    a haplotype of n genes, wih j randomly assigned to 1.";

MakeIndividuals[ψ : {FrequencyRepresentations[_, _] ...}, n_List,
   opts___Rule] :=
  MapThread[MakeIndividuals[#1, #2, opts] &, {ψ, n}];

MakeIndividuals[AlleleFrequencies[p_, _], n_, opts___Rule] :=
  HaploidIndividuals[Table[(If[RandomReal[] < #1, 1, 0] &) /@ p, {n}], n];
MakeIndividuals[HaploidFrequencies[ψ_, _], n_, opts___Rule] :=
  HaploidIndividuals[
   RandomList[n, ψ, HaploidTypes[Log[2, Length[ψ]], opts]], n];
MakeIndividuals[DiploidFrequencies[ψ_, _], n_, opts___Rule] :=
  Module[{ht = HaploidTypes[Log[2, Length[ψ]], opts]},
   DiploidIndividuals[RandomList[n, Flatten[ψ],
     Flatten[Outer[List, ht, ht, 1], 1]], n]];
MakeIndividuals[SymmetricHaploidFrequencies[ψ_, _], n_, opts___Rule] :=
  Module[{ng = Length[ψ] - 1},
   HaploidIndividuals[(MakeIndividuals[#1, ng] &) /@
     RandomList[n, ψ, Range[0, ng]], n]];
MakeIndividuals[SymmetricDiploidFrequencies[ψ_, _], n_, opts___Rule] :=
  Module[{ng = Length[ψ] - 1},
   DiploidIndividuals[Map[MakeIndividuals[#1, ng] &,
     ({Quotient[#1 - 1, ng + 1], Mod[#1 - 1, ng + 1]} &) /@
      RandomList[n, Flatten[ψ]], {2}], n]];
MakeIndividuals[SymmetricNeutralHaploidFrequencies[{ψ_, u_}, n0_],
   n_, opts___Rule] := Module[{ng = Length[ψ] - 1},
   HaploidIndividuals[
    (Append[MakeIndividuals[#1, ng], If[RandomReal[] < u[[#1 + 1]], 1, 0]] &) /@
     RandomList[n, ψ, Range[0, ng]], n]];
MakeIndividuals[SymmetricNeutralDiploidFrequencies[{ψ_, u_}, n0_],
   n_, opts___Rule] := Module[{ng = Length[ψ] - 1, fu = Flatten[u]},
   DiploidIndividuals[
    ({Append[MakeIndividuals[#1[[1]], ng],
        If[RandomReal[] < fu[[#1[[3]]]], 1, 0]],
       Append[MakeIndividuals[#1[[2]], ng],
        If[RandomReal[] < fu[[#1[[3]]]], 1, 0]]} &) /@
     ({Quotient[#1 - 1, ng + 1], Mod[#1 - 1, ng + 1], fu[[#1]]} &) /@
      RandomList[n, Flatten[ψ]], n]];
```

```
MakeIndividuals[j_Integer, n_Integer] :=
  Muddle[Join[Array[1 &, j], Array[0 &, n - j]]];
```

MakeOptionsHaploid

```
MakeOptionsHaploid::usage =
  "MakeOptionsHaploid is a rule which converts options such as
    NumericalModel->DiploidFrequencies to their haploid version";
```

MakePopulation

```
MakePopulation::usage =
  "MakePopulation[nloci,p] gives a haploid population at linkage
    equilibrium, with allele frequency p; this is represented
    as HaploidFrequencies[{ψ000…}]. The option NumericalModel
    can be used to give other representations.  For example,
    MakePopulation[nloci,{p,u},NumericalModel->
    SymmetricNeutralHaploidFrequencies gives a haploid
    population at LE, with selected loci at frequency p, and
    a single neutral marker at frequency u.
    MakePopulation[nloci,p,NumericalModel->HaploidIndividuals]
    gives a single haploid individual, drawn at random.
    MakePopulation[nloci,p,ndeme,opts] allows deme size to
    be specified; the default is 1.  ";

Options[MakePopulation] =
  {NumericalModel → HaploidFrequencies, SortByClass → False};

MakePopulation[nloci_Integer, p_, opts___Rule] :=
  MakePopulation[nloci, p, 1, opts];
```

```
MakePopulation[n_Integer, p_, ninds_, opts___Rule] :=
  Module[{i, nm = NumericalModel /. {opts} /. Options[MakePopulation]},
   Switch[nm,
    HaploidRepresentations,
     Switch[nm,
      HaploidFrequencies,
       HaploidFrequencies[
        HaploidTypes[n, opts] /.
        {{a___Integer} :> Times @@ ({a} (2 p - 1) + 1 - p)}, ninds],
      SymmetricHaploidFrequencies,
       SymmetricHaploidFrequencies[
        If[p == 0,
         ReplacePart[Array[0 &, n + 1], 1, 1],
          If[p == 1,
         ReplacePart[Array[0 &, n + 1], 1, -1],
            Table[Binomial[n, i] p^i (1 - p)^(n-i), {i, 0, n}]]], ninds],
      SymmetricNeutralHaploidFrequencies,
       SymmetricNeutralHaploidFrequencies[

       {MakePopulation[n, p[[1]], NumericalModel →
           SymmetricHaploidFrequencies][[1]], Array[p[[2]] &, n + 1]}, ninds],
      AlleleFrequencies,
       AlleleFrequencies[Array[1 &, n] p, ninds],
      HaploidIndividuals,
       MakeIndividuals[ AlleleFrequencies[Array[1 &, n] p, 1], ninds]],
     DiploidRepresentations,
      Switch[nm,
      DiploidIndividuals,
       DiploidIndividuals[Transpose[{
          MakePopulation[n, p, ninds, NumericalModel → HaploidIndividuals][[1]],
          MakePopulation[n, p, ninds, NumericalModel → HaploidIndividuals][[
           1]]}], ninds],
       _,
         MakeDiploid[MakePopulation[n, p, ninds, ##] & @@
        ({opts} /. MakeOptionsHaploid)]],
     _,
     (Message[NumericalModel::badSetting, nm]; {})]];
```

Mendel

```
Mendel::usage =
  "Mendel[{{0,1,…},{1,0,…}}] gives a random gamete, assuming
    no linkage.";

Mendel[pp : {{___}, {___}}] := Mendel /@ Transpose[pp];

Mendel[{x_, y_}] := If[RandomReal[Integer] == 0, x, y]
```

MeanW

```
MeanW::usage =
  "MeanW[ψ,W] gives the mean fitness of population ψ; W is the
    fitness function. MeanW[ψ,{{{SubscriptBox[W,0,  0,
    0],…}}}] gives the mean fitness, based on an appropriate
    table of fitnesses.";

AlleleFrequencies::badW = "W[p]=`1` does not have the form {W̄,{s₁…}}";

MeanW[ψ : {PopulationRepresentations[_, _] ...}, W_?(Not[ListQ[#]] &),
   opts___Rule] :=
  MapIndexed[MeanW[#1, W[First[#2]], opts] &, ψ];
MeanW[ψ : {PopulationRepresentations[_, _] ...}, W_List, opts___Rule] :=
  MapIndexed[MeanW[#1, W⟦First[#2]⟧, opts] &, ψ];

MeanW[SymmetricNeutralHaploidFrequencies[{ψ_, u_}, n_], W_,
   opts___Rule] :=
  MeanW[SymmetricHaploidFrequencies[ψ, n], W, opts];
MeanW[SymmetricNeutralDiploidFrequencies[{ψ_, u_}, _], W_,
   opts___Rule] :=
  MeanW[SymmetricDiploidFrequencies[ψ, n], W, opts];

MeanW[AlleleFrequencies[p_, _], W_?(Not[ListQ[#]] &), opts___Rule] :=
  Module[{wl = W[p]},
   If[MatchQ[wl, {_, _List}],
    First[wl],
    Message[AlleleFrequencies::badW, wl]; Indeterminate]];
MeanW[(fr : FrequencyRepresentations)[ψ_, _], W_?(Not[ListQ[#]] &),
   opts___Rule] :=
  Module[{n = Switch[fr,
      SymmetricRepresentations, Length[ψ] - 1,
      _, Log[2, Length[ψ]]]},
   Plus @@ Plus @@ (ψ * FitnessTable[W, n, NumericalModel → fr, opts])];
MeanW[HaploidIndividuals[ψ_, _], W_?(Not[ListQ[#]] &), opts___Rule] :=
  Mean[W /@ ψ];
MeanW[DiploidIndividuals[ψ_, _], W_?(Not[ListQ[#]] &), opts___Rule] :=
  Mean[W @@ # & /@ ψ];
```

```
MeanW[AlleleFrequencies[p_, _], W_List, opts___Rule] :=
  (Message[FitnessTable::badOption]; 1);
MeanW[FrequencyRepresentations[ψ_, _], W_List, opts___Rule] :=
  Plus @@ Plus @@ (ψ * W);
MeanW[HaploidIndividuals[ψ_, _], W_List, opts___Rule] :=
  Module[{ht},
   If[Length[ψ] == 0,
    Indeterminate,
    ht = HaploidTypes[Length[First[ψ]], opts];
    Mean[Extract[W, #] & /@ (First[Position[ht, #]] & /@ ψ)]]];
MeanW[DiploidIndividuals[ψ_, _], W_List, opts___Rule] :=
  Module[{ht},
   If[Length[ψ] == 0,
    Indeterminate,
    ht = HaploidTypes[Length[ψ[[1, 1]]], opts];
    Mean[Extract[W, #] & /@
      (Flatten[{Position[ht, #[[1]]], Position[ht, #[[2]]]}] & /@ ψ)]]];
```

MidParent

```
MidParent::usage =
  "MidParent[u] gives the matrix of mid-parental frequencies, (u_i+u_j)/2";

MidParent[u_] := Outer[Plus, u, u] / 2;
```

MigrateExact

```
MigrateExact::usage =
  "MigrateExact[ψ,ψ_s,m] implements migration from a source
    population ψ_s; ψ can be any valid representation of a
    population.  MigrateExact[{ψ_1,…},m] implements migration
    along a linear cline. The option FixedEnds→{ψ0, ψ1} allows
    fixed endpoints.  Where individuals are involved, each
    has an independent probability of migrating; for a single
    population, the parameter m is now the number of individuals
    immigrating from the source population, and must be an
    integer no greater than the population size.  FixedEnds
    must now specify a source with given HaploidFrequencies[]
    or DiploidFrequencies[].";

Options[MigrateExact] = {FixedEnds → {{}, {}}};
```

```
MigrateExact[(pr : NeutralRepresentations)[{ψ_, u_}, n_],
   (pr : NeutralRepresentations)[{ψs_, us_}, ns_], m_, opts___Rule] :=
  pr[{DivideZero[(1 - m) (n ψ) + m (ns ψs), (1 - m) n + m ns],
    DivideZero[(1 - m) (n u ψ) + m (ns us ψs), (1 - m) (n ψ) + m (ns ψs)]},
   (1 - m) n + m ns];
MigrateExact[(pr : FrequencyRepresentations)[ψ_, n_],
   (pr : PopulationRepresentations)[ψs_, ns_], m_, opts___Rule] :=
  pr[DivideZero[(1 - m) (n ψ) + m (ns ψs), (1 - m) n + m ns], (1 - m) n + m ns];
MigrateExact[(ir : IndividualRepresentations)[ψ_, n_],
   (fr : FrequencyRepresentations)[ψs_, ns_], Nm_Integer, opts___Rule] :=
  Module[{nStay = RandomReal[BinomialDistribution[n, (1 - Nm/n)]], np},
   np =
    ir[Muddle[Join[MakeIndividuals[fr[ψs, ns], Nm][[1]],
       RandomSet[nStay, ψ]]], nStay + Nm];
   If[MatchQ[ir, HaploidIndividuals] ∧ OddQ[nStay + Nm],
    HaploidIndividuals[Drop[np[[1]], 1], np[[2]] - 1], np]];

MigrateExact[ϕ : {(pr : NeutralRepresentations)[_, _] ..}, m_,
   opts___Rule] :=
  Module[{pp, os},
      os = FixedEnds /. {opts} /. Options[MigrateExact];
      If[os[[1]] === {} ⋁ os[[1]] === {{}}, os[[1]] = First[ϕ]];
      If[os[[-1]] === {} ⋁ os[[-1]] === {{}}, os[[-1]] = Last[ϕ]];
   pp = Prepend[Append[ϕ, Last[os]], First[os]];
   pp = pp /. {NeutralRepresentations[{ψ_, u_}, n_] :↦ {n ψ, n ψ u, n}};
   pr[{DivideZero[#[[1]], #[[3]]], DivideZero[#[[2]], #[[1]]]}, #[[3]] & /@
    (Drop[Drop[(1 - m) pp + m/2 (RotateLeft[pp] + RotateRight[pp]), 1], -1])];

MigrateExact[ϕ : {(pr : FrequencyRepresentations)[_, _] ..}, m_,
   opts___Rule] :=
  Module[{pp, os},
      os = FixedEnds /. {opts} /. Options[MigrateExact];
      If[os[[1]] === {} ⋁ os[[1]] === {{}}, os[[1]] = First[ϕ]];
      If[os[[-1]] === {} ⋁ os[[-1]] === {{}}, os[[-1]] = Last[ϕ]];
   pp = Prepend[Append[ϕ, Last[os]], First[os]];
   pp = pp /. {FrequencyRepresentations[ψ_, n_] :↦ {n ψ, n}};
   pr[DivideZero[#[[1]], #[[2]]], #[[2]] & /@
    (Drop[Drop[(1 - m) pp + m/2 (RotateLeft[pp] + RotateRight[pp]), 1], -1])];
```

```
MigrateExact[ϕ : {(ir : IndividualRepresentations)[_, _] ..}, m_,
   opts___] :=
  Module[{pp, os, nd = Length[ϕ], j, res, mig, migleft, migright,
    leftFree, rightFree, leftImm, rightImm, npp, np,
    hQ = MatchQ[ir, HaploidIndividuals]},
   os = FixedEnds /. {opts} /. Options[MigrateExact];
   leftFree = (os[[ 1]] === {} ⋁ os[[ 1]] === {{}});
   rightFree = (os[[-1]] === {} ⋁ os[[-1]] === {{}});
   pp = ϕ /. {ir[ψ_, n_] ⇒ ψ};
   npp = Table[
      res = RandomSet[RandomReal[BinomialDistribution[Length[pp[[j]]],
          (1 - m)]], pp[[j]]];
      mig = TakeAway[pp[[j]], res];
      migleft =
       RandomSet[RandomReal[BinomialDistribution[Length[mig], 1/2]], mig];
      migright = TakeAway[mig, migleft];
      {migleft, res, migright}, {j, nd}] // Transpose;
   leftImm = If[leftFree, npp[[1, 1]],
     MakeIndividuals[os[[1]], Length[npp[[1, 1]]][[1]]];
   rightImm = If[rightFree, npp[[3, -1]],
     MakeIndividuals[os[[-1]], Length[npp[[3, -1]]][[1]]];
   npp = ReplacePart[ReplacePart[npp, leftImm, {3, -1}], rightImm,
     {1, 1}];
   If[hQ ⋀ OddQ[Length[#]], ir[Drop[Muddle[#], 1], Length[#] - 1],
      ir[Muddle[#], Length[#]]] & /@
    Apply[Join, ({RotateLeft[npp[[1]]], npp[[2]], RotateRight[npp[[3]]]} //
       Transpose), {1}]];
```

Mutate

```
Mutate::usage =
  "Mutate[ψ,{{μ,ν},…}] mutates with probability μ from 0 to 1,
    and ν from 1 to 0.  Compiled->True is much faster. When
    applied to FrequencyRepresentations, uses MutationMatrix[]
    to store the transformation matrix. For the symmetric
    model,Mutate[SymmetricHaploidFrequencies[],{μ,ν}] should
    be used, because mutation rates must be the same across loci.";

Options[Mutate] = {Compiled → False};

Mutate[g_List, mr_List, opts___Rule] :=
  If[Compiled /. {opts} /. Options[Mutate],
   MutateCompiled[HaploidIndividuals][g, mr],
   MapThread[If[RandomReal[] < #2[[#1 + 1]], 1 - #1, #1] &, {g, mr}]];

Mutate[HaploidIndividuals[ψ_, n_], mr_List, opts___Rule] :=
  HaploidIndividuals[Mutate[#, mr, opts] & /@ ψ, n];
```

```
Mutate[AlleleFrequencies[p_, n_], mr_List, opts___Rule] :=
  AlleleFrequencies[
   If[Compiled /. {opts} /. Options[Mutate],
    MutateCompiled[AlleleFrequencies][p, mr],
    Module[{tmr = Transpose[mr]},
     p (1 - tmr[[1]] - tmr[[2]]) + tmr[[1]]]], n];

Mutate[HaploidFrequencies[ψ_, n_], mr_List, opts___Rule] :=
  HaploidFrequencies[ConstructMutateMatrix[mr, Log[2, Length[ψ]], opts].
    ψ, n];

Mutate[SymmetricHaploidFrequencies[ψ_, n_], mr_List, opts___Rule] :=
  SymmetricHaploidFrequencies[
   ConstructMutateMatrix[mr, Length[ψ] - 1, opts,
     NumericalModel → SymmetricHaploidFrequencies].ψ, n];

Mutate[SymmetricNeutralHaploidFrequencies[ψ_, n_], mr_List, opts___Rule] :=
  0;
```

MutateCompiled

```
MutateCompiled::usage =
  "MutateCompiled[HaploidFrequencies][{SubscriptBox[p,{0,
    …}],…},{{μ,ν},…}] alters haploid genotype frequencies to
    allow for mutation.
    MutateCompiled[SymmetricHaploidFrequencies][{p0,…},{μ,ν}]
    applies to the symmetric model; similarly for
    SymmetricNeutralFrequencies. These versions use
    ConstructMutateMatrix[] to store a large matrix.
    MutateCompiled[AlleleFrequencies][{p1,…},{{μ,ν},…}] applies
    to allele frequencies.
    MutateCompiled[HaploidIndividuals][{0,1,…},{{μ,ν},…}]
    mutates a genotype {0,1,…} with probability μ from 0 to
    1, and ν from 1 to 0.  For short genomes, MutateCompiled[]
    is faster than using Mutate[…,Compiled->True], because
    there is no need to check options.";

MutateCompiled[HaploidIndividuals] =
  Compile[{{g, _Integer, 1}, {mr, _Real, 2}},
   Module[{j}, Table[If[RandomReal[] < mr[[j, g[[j]] + 1]], 1 - g[[j]], g[[j]]],
     {j, Length[g]}]], {{j, _Integer}}];

MutateCompiled[AlleleFrequencies] =
  Compile[{{p, _Real, 1}, {mr, _Real, 2}},
   Module[{tmr = Transpose[mr]},
    p (1 - tmr[[1]] - tmr[[2]]) + tmr[[1]]]];

MutateCompiled[HaploidFrequencies][ψ_, mr_, opts___Rule] :=
  ConstructMutateMatrix[mr, Log[2, Length[ψ]], opts, Compiled → True].ψ;

MutateCompiled[SymmetricHaploidFrequencies][ψ_, mr_] :=
  ConstructMutateMatrix[mr, Length[ψ] - 1, Compiled → True,
    NumericalModel → SymmetricHaploidFrequencies].ψ;
```

Mutation

```
Mutation::usage =
  "Mutation->{{μ,ν},…} is an option for NewGametes which specifies
    mutation at rate μ from 0 to 1, and ν from 1 to 0.
    Mutation->{{μ},…} is shorthand for a rate μ in each
    direction.  Mutation->{μ,ν} or {μ} specifies the same
    rates at all loci.";
```

NetRecombination[{r1, r2, r3, …}] gives the net recombination rate across a linear genetic map

```
NetRecombination::usage =
  "NetRecombination[{r1,r2,r3,…}] gives the net recombination
    rate across a linear genetic map";

NetRecombination[{}] := 0;
NetRecombination[{r___, s_}] := s + NetRecombination[{r}] (1 - 2 s);
```

NeutralRepresentations

```
NeutralRepresentations::usage =
  "NeutralRepresentations is a pattern which matches
    SymmetricNeutralHaploidFrequencies|
    SymmetricNeutralDiploidFrequencies.";

NeutralRepresentations = SymmetricNeutralHaploidFrequencies |
   SymmetricNeutralDiploidFrequencies;
```

NewGametes

```
NewGametes::usage =
  "NewGametes[ψ] gives the haploid population after meiosis; ψ
    must represent a diploid population, or a list of diploid
    populations.  By default, deme size is doubled.
    NewGametes[ψ,W] does the same, but assuming fitness
    function W.  DemeSize-> f[n,W̄] allows for changes in deme
    size, to 2f[n,W̄].  Lligam→{SubscriptBox[\(r\),\(1,\(\(\\)\(
    \)\(\(\(\\)\( \)\)\(\(\(\(\\)\(
    \)\)\)2\)\)\)\)],SubscriptBox[\(r\),\(2,\(\(\\)\(
    \)\(\(\(\\)\( \)\)\(\(\(\(\\)\( \)\)\)3\)\)\)\)],…} can
    be used to specify a linear genetic map, where appropriate.
    RandomDrift->True samples haploid gametes from the diploid
    population; does not apply to symmetric or individual
    representations. TruncationSelection->True applies with
    individual representations, and takes the f[n,W̄] individuals
    with highest W as parents.  Gametes are then sampled
    randomly from these parents, so not all the chosen parents
    may actually reproduce.  Mutation->{{μ,ν},…} allows for
    mutation at rate μ from allele 0 to 1, and ν from 1 to 0.";

Options[NewGametes] = {Lligam → False, Mutation → None, DemeSize → (#1 &),
   RandomDrift → False, TruncationSelection → False};
```

```
NewGametes::badN = "New deme size `1` is greater than original size `2`";

NewGametes[AlleleFrequencies[p_, n_], opts___Rule] :=
  NewGametes[AlleleFrequencies[p, n], {1, Array[0 &, Length[p]]} &, opts];
NewGametes[(pr : DiploidRepresentations)[ψ_, n_], opts___Rule] :=
  NewGametes[pr[ψ, n], 1 &, opts];

NewGametes[AlleleFrequencies[p_, n_], W_, opts___Rule] :=
  Module[{f = DemeSize /. {opts} /. Options[NewGametes], ww = W[p],
    rd = RandomDrift /. {opts} /. Options[NewGametes], pn,
    mut = Mutation /. {opts} /. Options[NewGametes]},
   f = f /. {r_, β_, γ_, λ_} → ExponentialF[r, β, γ, λ];
   pn = AlleleFrequencies[p (1 + (1 - p) ww[[2]]/ww[[1]]), 2 f[n, ww[[1]]]];
   If[rd, pn = RandomDrift[pn, pn[[2]]]];
   If[mut === None,
    pn,
    Mutate[pn, ConstructMutateList[mut, Length[pn]], opts]]];

NewGametes[DiploidFrequencies[ϕ_, n_], W_, opts___Rule] :=
  Module[{f = DemeSize /. {opts} /. Options[NewGametes],
    wb = MeanW[DiploidFrequencies[ϕ, n], W, opts],
    rd = RandomDrift /. {opts} /. Options[NewGametes], hn,
    mut = Mutation /. {opts} /. Options[NewGametes]},
   f = f /. {r_, β_, γ_, λ_} → ExponentialF[r, β, γ, λ];
   hn = HaploidFrequencies[If[Length[ϕ] == 0, {},
      If[ϕ[[1]] === {},
       {},
       Plus @@ Plus @@ (GameteTable[Log[2, Length[ϕ]], W, opts] * ϕ) / wb]],
     2 f[n, wb]];
   If[rd, hn = RandomDrift[hn, hn[[2]]]];
   If[mut === None,
    hn,
    Mutate[hn, ConstructMutateList[mut, Log[2, Length[ϕ]]], opts]]];
```

```
NewGametes[SymmetricDiploidFrequencies[ϕ_, n_], W_, opts___Rule] :=
  Module[{f = DemeSize /. {opts} /. Options[NewGametes],
    wb = MeanW[SymmetricDiploidFrequencies[ϕ, n], W, opts],
    mut = Mutation /. {opts} /. Options[NewGametes],
    ng, nn},
   f = f /. {r_, β_, γ_, λ_} → ExponentialF[r, β, γ, λ];
   nn = 2 f[n, wb];
   If[Length[ϕ] == 0,
    SymmetricHaploidFrequencies[{}, nn],
    If[ϕ[[1]] === {},
     SymmetricHaploidFrequencies[{}, nn],
     ng = SymmetricHaploidFrequencies[
       Plus @@
         Plus @@ (GameteTable[Length[ϕ] - 1, W,
             NumericalModel → SymmetricDiploidFrequencies] * ϕ) / wb, nn];
     If[mut === None,
      ng,
      Mutate[ng, mut /. {μ_} → {μ, μ}, opts]]]];

NewGametes[SymmetricNeutralDiploidFrequencies[{ϕ_, u_}, n_], W_,
   opts___Rule] :=
  Module[{f = DemeSize /. {opts} /. Options[NewGametes],
    wb = MeanW[SymmetricNeutralDiploidFrequencies[ϕ, n], W, opts],
    gt, nψ, nψu},
   f = f /. {r_, β_, γ_, λ_} → ExponentialF[r, β, γ, λ];
   SymmetricNeutralHaploidFrequencies[If[Length[ϕ] == 0, {},
     If[ϕ[[1]] === {},
      {},
      gt = GameteTable[Length[ϕ] - 1, W,
        NumericalModel → SymmetricNeutralDiploidFrequencies];
      nψ = Plus @@ Plus @@ (gt * ϕ);
      nψu = Plus @@ Plus @@ (gt * ϕ * u);
      {nψ / wb, DivideZero[nψu, nψ]}]],
    2 f[n, wb]]];
```

```
NewGametes[ϕ : DiploidIndividuals[ψ_, n_], W_, opts___Rule] :=
  Module[{f = DemeSize /. {opts} /. Options[NewGametes],
    wb = MeanW[ϕ, W, opts], wl, newN, rl, ht, tp, np, ss,
    ts = TruncationSelection /. {opts} /. Options[NewGametes],
    mut = Mutation /. {opts} /. Options[NewGametes]},
   wl = If[ListQ[W],
     ht = HaploidTypes[Length[ψ[[1, 1]]], opts];
     (Extract[W, #] & /@
       (Flatten[{Position[ht, #[[1]]], Position[ht, #[[2]]]}] & /@ ψ)),
     W @@ # & /@ ψ];
   f = f /. {r_, β_, γ_, λ_} → ExponentialF[r, β, γ, λ];
   newN = 2 Round[f[n, wb]];
   rl = If[ts,
     If[newN > n,
      Message[NewGametes::badN, newN, n]; newN = n];
     ss = Transpose[{wl, Range[n]}];
     tp = Sort[wl][[-newN]]; np = Select[ss, (#[[1]] > tp) &];
     np = Join[np, RandomSet[newN - Length[np],
        Select[ss, (#[[1]] == tp) &]]];
     Muddle[Last /@ np],
     RandomList[newN, wl / Plus @@ wl]];
   If[mut === None,
    HaploidIndividuals[RandomFamily[ψ[[#]], opts] & /@ rl, newN],
    HaploidIndividuals[
     MutateCompiled[HaploidIndividuals][RandomFamily[ψ[[#]], opts],
        ConstructMutateList[mut, Length[ψ[[1, 1]]]]] & /@ rl, newN]]];
```

NonNeutralRepresentations

```
NonNeutralRepresentations::usage =
  "NonNeutralRepresentations is a pattern which matches any
    valid representation except NeutralRepresentations.";

NonNeutralRepresentations =
  HaploidFrequencies | DiploidFrequencies | AlleleFrequencies |
   SymmetricHaploidFrequencies | SymmetricDiploidFrequencies |
   HaploidIndividuals | DiploidIndividuals;
```

NonSymmetricRepresentations

```
NonSymmetricRepresentations::usage =
  "NonSymmetricRepresentations is a pattern which matches any
    valid representation except SymmetricRepresentations or
    AlleleFrequencies.";

NonSymmetricRepresentations =
  HaploidFrequencies | DiploidFrequencies | HaploidIndividuals |
   DiploidIndividuals;
```

NumberOfGenes

```
NumberOfGenes::usage =
  "NumberOfGenes[ψ] gives the number of genes, where ψ represents
    a population.  Also applies to a list of ψ's, representing
    a cline.";

NumberOfGenes::noIndividuals = "No individuals in this population";

NumberOfGenes[{(pr : PopulationRepresentations)[ψ_, n_], ___}] :=
  NumberOfGenes[pr[ψ, n]];

NumberOfGenes[(pr : PopulationRepresentations)[ψ_, _]] :=
  Switch[pr,
   HaploidFrequencies | DiploidFrequencies,
   Log[2, Length[ψ]],
   AlleleFrequencies,
   Length[ψ],
   SymmetricHaploidFrequencies | SymmetricDiploidFrequencies,
   Length[ψ] - 1,
   SymmetricNeutralHaploidFrequencies |
    SymmetricNeutralDiploidFrequencies,
   Length[ψ[[1]]] - 1,
   HaploidIndividuals,
   If[ψ == {}, Message[NumberOfGenes::noIndividuals], Length[ψ[[1]]]],
   DiploidIndividuals,
   If[ψ == {}, Message[NumberOfGenes::noIndividuals], Length[ψ[[1, 1]]]]];
```

NumericalModel

```
NumericalModel::usage =
  "NumericalModel is an option which determines how a population
    is represented.  Possible values are any
    PopulationRepresentations";

NumericalModel::badSetting = "Invalid setting `1` for NumericalModel";
```

Parents

```
Parents::usage =
  "Parents[P] returns the parents of each individual in the
    pedigree.  P must be a SparseArray.";

Parents[P_SparseArray] :=
  Partition[
   #[[1, 2]] & /@
    Drop[ArrayRules[P] /. {(x_ -> 2) -> Sequence[x -> 2, x -> 2]}, -1],
   2];
```

PopulationRepresentations

```
PopulationRepresentations::usage =
  "PopulationRepresentations is a pattern which matches any
    valid representation of a population.";
```

```
PopulationRepresentations =
  HaploidFrequencies | DiploidFrequencies | AlleleFrequencies |
   SymmetricHaploidFrequencies | SymmetricDiploidFrequencies |
   SymmetricNeutralHaploidFrequencies |
   SymmetricNeutralDiploidFrequencies | HaploidIndividuals |
   DiploidIndividuals;
```

PowerF

```
PowerF::usage =
  "PowerF[r,β,γ][n,w̄] is the function rSuperscriptBox[n,1 - γ]
    w̄^β. DemeSize->PowerF[r,β,γ] can be passed to compiled
    versions of IterateExact and StoreExact.  PowerF[n*,0,1][n,w̄]
    represents a fixed number n*.  PowerF[r,β,γ,λ][n,w̄]
    represents the generalised form rSuperscriptBox[n,1 - γ]
    w̄^β Exp[-λn].  PowerF[r…][j][n,w̄] is the same as PowerF[r…][n,w̄].";

PowerF[r_, β_, γ_][n_, wb_] := r n^(1-γ) wb^β;
PowerF[r_, β_, γ_, λ_][n_, wb_] := r n^(1-γ) wb^β Exp[-λn];

PowerF[r_, β_, γ_][_][n_, wb_] := r n^(1-γ) wb^β;
PowerF[r_, β_, γ_, λ_][_][n_, wb_] := r n^(1-γ) wb^β Exp[-λn];
```

MigrationOrder

```
MigrationOrder::usage =
  "MigrationOrder is a option for IterateExact which determines
    the order of the life cycle.  Can take values UMS, MUS, or USM,";
```

UMS

```
UMS::usage =
  "UMS is a setting for MigrationOrder, which represents the
    order random union→migration→selection→meiosis in IterateExact.";
```

MUS

```
MUS::usage =
  "MUS is a setting for MigrationOrder, which represents the
    order migration→random union→selection→meiosis in IterateExact.";
```

USM

```
USM::usage =
  "USM is a setting for MigrationOrder, which represents the
    order random union→selection→migration→meiosis in IterateExact.";
```

RandomDrift

```
RandomDrift::usage =
  "RandomDrift[ψ,n] gives the random frequencies generated by
    sampling n individuals from the genotype or allele
    frequencies ψ.  Compiled->True uses compiled code. Deme
    size is changed to n.  RandomDrift[ψ,∞] makes no change. ";
```

```
RandomDrift::symmetric =
  "Random drift is not allowed under the symmetric model";

RandomDrift::individuals =
  "RandomDrift does not apply to populations of individuals.";

RandomDrift[ψ : {PopulationRepresentations[_, _] ...}, n_List] :=
  MapThread[RandomDrift[#1, #2] &, {ψ, n}];

RandomDrift[(pr : PopulationRepresentations)[ψ_, n_], ∞] :=
  pr[ψ, n];
RandomDrift[(pr : IndividualRepresentations)[ψ_, n_], _] :=
  (Message[RandomDrift::individuals]; pr[ψ, n]);
RandomDrift[(pr : SymmetricRepresentations)[ψ_, n_], _] :=
  (Message[RandomDrift::symmetric]; pr[ψ, n]);
RandomDrift[AlleleFrequencies[p_, n_], ns_] :=
  Module[{nsr = Max[1, Round[ns]]},
   AlleleFrequencies[Last[RandomMultinomial[nsr, {1 - #, #}]] / N[nsr] & /@ p,
    ns]];
RandomDrift[HaploidFrequencies[ψ_, n_], ns_] :=
  Module[{nsr = Max[1, Round[ns]]},
   HaploidFrequencies[RandomMultinomial[nsr, ψ] / N[nsr], ns]];
RandomDrift[DiploidFrequencies[ψ_, n_], ns_] :=
  Module[{nsr = Max[1, Round[ns]]},
   DiploidFrequencies[
    Partition[RandomMultinomial[nsr, Flatten[ψ]], Length[ψ]] / N[nsr], ns]];
```

RandomFamily[p1, p2] gives a random offspring from a cross between genotypes p1, p2.

```
RandomFamily::usage =
  "RandomFamily[p] gives a haploid offspring from the diploid
    parent p.  RandomFamily[p1,p2] gives a random diploid
    offspring from a cross between diploid genotypes p1, p2.
    RandomFamily[p1,p2,n] generates n such offspring.
    Linkage→{r12,r23,…} allows for linkage.
    SegregationRatio→{0.6,0.7,…} allows for a biased segregation
    at each locus; the proportion of alleles derived from the
    first genome in the diploid is given.";

Options[RandomFamily] = {Lligam → False, SegregationRatio → False};

RandomFamily[p__List, n_Integer, opts___Rule] :=
  Table[RandomFamily[p, opts], {n}];

RandomFamily[p1_List, p2_List] := {RandomFamily[p1], RandomFamily[p2]};
```

```
RandomFamily[p1_List, p2_List, opts__Rule] :=
  {RandomFamily[p1,
    opts /. {(Lligam → {x_List, y_List}) :→ (Lligam → x),
      (SegregationRatio → {x_List, y_List}) :→ (SegregationRatio → x)}],
   RandomFamily[p2,
    opts /. {(Lligam → {x_List, y_List}) :→ (Lligam → y),
      (SegregationRatio → {x_List, y_List}) :→ (SegregationRatio → y)}]};

RandomFamily[{h1_List, h2_List}, opts___Rule] :=
  Module[{
    rl = Lligam /. {opts} /. Options[RandomFamily],
    sr = SegregationRatio /. {opts} /. Options[RandomFamily],
    x, y, tss = Transpose[{h1, h2}], j},
   If[rl === False,
    If[sr === False, sr = Table[1/2, {Length[h1]}]];
    MapThread[(#1[[If[RandomReal[] < #2, 1, 2]]]) &,
     {Transpose[{h1, h2}], sr}],
    x = RandomReal[Integer, {1, 2}]; y = {tss[[1, x]]};
    Do[If[RandomReal[] < rl[[j]], x = If[x == 1, 2, 1]];
     AppendTo[y, tss[[j + 1, x]]], {j, Length[rl]}]; y]];
```

- **SegregationRatio is an option for RandomFamily that specifies the probability that the allele from the first genome in the diploid is transmitted. SegregationRatio → {0.6, 0.7, …} specifies different ratios for each locus; SegregationRatio → 1/2, the default, specifies the same ratio at all loci.**

```
SegregationRatio::usage =
  "SegregationRatio is an option for RandomFamily that specifies
    the probability that the allele from the first genome in
    the diploid is transmitted.  SegregationRatio→{0.6,0.7,…}
    specifies different ratios for each locus.
    SegregationRatio→{{0.6,0.7,…},{0.8,…}} allows different
    ratios in the two parents. SegregationRatio→1/2, the
    default, specifies the same ratio at all loci.";
```

- **SortByClass is an option for HaploidTypes[] and Diploid Types[] which determines the order of the genotypes.**

```
SortByClass::usage =
  "SortByClass is an option for HaploidTypes[] and Diploid
    Types[] which determines the order of the genotypes.";
```

SparsePedigreeMatrix

```
SparsePedigreeMatrix::usage =
  "SparsePedigreeMatrix[n] gives a random matrix.  SparsePedigreeMatrix[z,Va]
     allows for selection according to the infinitesimal model; returns {P,z}
     where z are the trait values amongst offspring; the z must be Real.
     SparsePedigreeMatrix[p,w] takes a list of genotypes p (each {0,1,…})
     and generates {P,p} according to the fitness function w[{0,0,1,…}].";

SparsePedigreeMatrix[n_Integer] := Module[{j, p1, p2, tt},
   tt = Table[(p1 = RandomReal[Integer, {1, n}]; p2 = RandomReal[Integer, {1, n}];
       If[p1 == p2, {j, p1} → 2, {{j, p1} → 1, {j, p2} -> 1}]), {j, n}];
   SparseArray[Flatten[tt, 1], {n, n}]];

SparsePedigreeMatrix[z : {__Real}, v_] :=
  Module[{n = Length[z], j, tt, ww = Exp[z], pars, sd = Sqrt[v/2]},
   pars = Partition[RandomList[2 n, ww/Plus @@ ww, Compiled → True], 2];
   tt = Table[If[pars[[j, 1]] == pars[[j, 2]], {j, pars[[j, 1]]} → 2,
       {{j, pars[[j, 1]]} → 1, {j, pars[[j, 2]]} -> 1}], {j, n}];
   {SparseArray[Flatten[tt, 1], {n, n}],
    RandomReal[NormalDistribution[0, sd], n] + (Mean[z[[#]]] & /@ pars)}];

SparsePedigreeMatrix[p : {{__Integer} ...}, w_] :=
  Module[{n = Length[p], j, tt, pars, ww = w /@ p},
   pars = Partition[RandomList[2 n, ww/Plus @@ ww, Compiled → True], 2];
   tt = Table[If[pars[[j, 1]] == pars[[j, 2]], {j, pars[[j, 1]]} → 2,
       {{j, pars[[j, 1]]} → 1, {j, pars[[j, 2]]} -> 1}], {j, n}];
   {SparseArray[Flatten[tt, 1], {n, n}], Mendel[p[[#]] & /@ pars}];
```

SparsePedigreeMatrixDemes

```
SparsePedigreeMatrixDemes::usage =
  "SparsePedigreeMatrixDemes[n,d,m] gives a random matrix, with
    d demes of size n, and island model migration at rate m.";

SparsePedigreeMatrixDemes[n_Integer, nd_Integer, m_] :=
  Module[{j, d, jd, p1, p2, tt},
   tt = Table[(jd = j + (d - 1) n;
      p1 = RandomReal[Integer, {(d - 1) n + 1, n * d}];
      p2 = If[RandomReal[] < m, RandomReal[Integer, {1, n * nd}],
        RandomReal[Integer, {(d - 1) n + 1, n * d}]];
      If[p1 == p2, {jd, p1} → 2, {{jd, p1} → 1, {jd, p2} -> 1}]), {d, nd},
     {j, n}];
   SparseArray[Flatten[tt, 2], {n * nd, n * nd}]];
```

SparsePedigreeMatrixSelfed

```
SparsePedigreeMatrixSelfed::usage =
  "SparsePedigreeMatrixSelfed[n,α] gives a random matrix,
    allowing for selfing at a rate α.  Individuals are selfed
    with proability α.";

SparsePedigreeMatrixSelfed[n_Integer, α_] := Module[{j, p1, p2, tt},
   tt = Table[(p1 = RandomReal[Integer, {1, n}];
      p2 = If[RandomReal[] < α, p1, RandomReal[Integer, {1, n}]];
      If[p1 == p2, {j, p1} → 2, {{j, p1} → 1, {j, p2} -> 1}]), {j, n}];
   SparseArray[Flatten[tt, 1], {n, n}]];
```

SparsePedigreeMatrixStepping(*TO BE DEFINED*)

```
SparsePedigreeMatrixDemes::usage =
  "SparsePedigreeMatrixDemes[n,d,m] gives a random matrix, with
    d demes of size n, and island model migration at rate m.";

SparsePedigreeMatrixDemes[n_Integer, nd_Integer, m_] :=
  Module[{j, d, jd, p1, p2, tt},
   tt = Table[(jd = j + (d - 1) n;
      p1 = RandomReal[Integer, {(d - 1) n + 1, n * d}];
      p2 = If[RandomReal[] < m, RandomReal[Integer, {1, n * nd}],
        RandomReal[Integer, {(d - 1) n + 1, n * d}]];
      If[p1 == p2, {jd, p1} → 2, {{jd, p1} → 1, {jd, p2} -> 1}]), {d, nd},
     {j, n}];
   SparseArray[Flatten[tt, 2], {n * nd, n * nd}]];
```

StoreExact

```
StoreExact::usage =
  "StoreExact[ψ,ψ0,W,m,{t,dt}] stores results at intervals dt
    in ψ[t], starting with ψ0.  ψ0 can represent a single
    population or a cline.  The options Compiled, FixedEnds,
    DemeSize, RandomDrift and MigrationOrder can be used, as
    for IterateExact. NumericalModel specifies the representation.
    SymmetricCline→True simulates a symmetric cline more efficiently.";

Options[StoreExact] = {SymmetricCline → False, FixedEnds → {{}, {}}};

StoreExact::usage =
  "StoreExact[ψ,ψ0,W,m,{t,dt}] stores results at intervals dt
    in ψ[t], starting with ψ0.  ψ0 can represent a single
    population or a cline.  The options Compiled, FixedEnds,
    DemeSize and MigrationOrder can be used, as for IterateExact.
    NumericalModel specifies the representation. SymmetricCline→True
    simulates a symmetric cline more efficiently.";
```

```
StoreExact[ψ_, ψ0 : PopulationRepresentations[_, _], ww_, m_,
   {t_, dt_}, opts___Rule] :=
  Module[{tt = 0, fe = FixedEnds /. {opts} /. Options[StoreExact]},
   ψ[0] = ψ0;
   While[tt + dt < t,
    ψ[tt + dt] = Nest[IterateExact[#, ww, m, opts] &, ψ[tt], dt]; tt += dt];
    ψ[t] = Nest[IterateExact[#, ww, m, opts] &, ψ[tt], t - tt]];

StoreExact[ψ_, ψ0 : {PopulationRepresentations[_, _] ..}, ww_, m_,
   {t_, dt_}, opts___Rule] :=
  Module[{ψt, ff, tt = 0, nd2 = Ceiling[nd / 2], optsNF,
    eQ = EvenQ[Length[ψ0]],
    sQ = SymmetricCline /. {opts} /. Options[StoreExact],
    fe = FixedEnds /. {opts} /. Options[StoreExact], nfe},

   optsNF = Sequence @@ ({opts} /. {{a___, FixedEnds → _, b___} :> {a, b}});
    ψ[0] = ψ0;
   If[sQ,
    nfe = If[eQ, ψt[[-1]], ψt[[-2]]] /.
      {(pr : NeutralRepresentations)[{ϕ_List, u_List}, n_] :↦
       pr[{Reverse[ϕ], Reverse[u]}, n],
       (pr : NonNeutralRepresentations)[ϕ_List, n_] :↦ pr[Reverse[ϕ], n]};
    ff[g_] := (ψt = Take[g, nd2];
      DoubleCline[IterateExact[ψt, ww, m, optsNF, FixedEnds → {fe[[1]], nfe}],
       eQ]),
    ff[g_] := IterateExact[g, ww, m, opts]];
    While[tt + dt < t, ψ[tt + dt] = Nest[ff, ψ[tt], dt]; tt += dt];
    ψ[t] = Nest[ff, ψ[tt], t - tt]];

StoreNeutral[ψ_, ww_, m_, {t_, dt_}, {nd_, ng_, dp___}, opts___] :=
    Module[{ψt, ff, tt = 0, nd2 = Ceiling[(nd + 1) / 2], optsNF,
            eQ = EvenQ[nd],
            sQ = SymmetricCline /. {opts} /. Options[StoreExact],
            fe = FixedEnds /. {opts} /. Options[StoreExact]},
     optsNF = opts /. {{a___, FixedEnds → _, b___} :> {a, b}};
      ψ[0] = InitialiseDemes[nd, ng, dp, opts, Neutral → True];
       If[sQ,

    ff[g_] := (ψt = Take[g, nd2];
      DoubleCline[IterateNeutral[ψt, ww, m, optsNF,
        FixedEnds → {fe[[1]], Reverse /@ If[eQ, g[[-1]], g[[-2]]]}], eQ]),
            ff[g_] := IterateNeutral[g, ww, m, opts]];
    While[tt + dt < t, ψ[tt + dt] = Nest[ff, ψ[tt], dt]; tt += dt];
    ψ[t] = Nest[ff, ψ[tt], t - tt]];
```

StoreResults

```
StoreResults::usage =
  "StoreResults→False is an option for GameteTable[] which
    prevents results being stored.";
```

SymmetricCline

```
SymmetricCline::usage =
  "SymmetricCline is an option for StoreExact[] which allows a
    symmetrical cline to be simulated more efficiently.";
```

SymmetricDiploidFrequencies

```
SymmetricDiploidFrequencies::usage =
  "SymmetricDiploidFrequencies[{{SubscriptBox[ψ,0, 0],…},…},n]
    represents a diploid population in which haplotypes with
    the same # of '1' alleles are equally frequent.  This
    allows a diploid population to be repesented by (n+1)^2
                                                    2
    variables, rather than 2 .
    SymmetricDiploidFrequencies[{{SubscriptBox[ψ,0, 0],…},…}]
    is shorthand for SymmetricDiploidFrequencies[{{SubscriptBox[ψ,0,
    0],…},…},1]";

SymmetricDiploidFrequencies[ψ_List] := SymmetricDiploidFrequencies[ψ, 1];
```

SymmetricHaploidFrequencies

```
SymmetricHaploidFrequencies::usage =
  "SymmetricHaploidFrequencies[{ψ0,…},n] represents a haploid
    population in which genotypes with the same # of '1'
    alleles are equally frequent.  This allows a haploid
    population to be repesented by n+1 variables, rather than
    2^n. SymmetricHaploidFrequencies[{ψ0,…}] is shorthand for
    SymmetricHaploidFrequencies[{ψ0,…},1]";

SymmetricHaploidFrequencies[ψ_List] := SymmetricHaploidFrequencies[ψ, 1];
```

SymmetricMeanW

```
SymmetricMeanW::usage =
  "SymmetricMeanW[i,j,n,W] gives the mean fitness of the diploid
    combination {i,j}, averaging over heterozygosities; there
    are n genes.  Fitness is W[i,j,h].  SymmetricMeanW[n,W]
    gives a table of diploid fitnesses.
    SymmetricMeanW[n,{{{SubscriptBox[W,0,  0, 0],…}…}…}] gives
    the mean fitness of the diploid combination {i,j}, averaging
    over heterozygosities; there are n genes.  Fitness is
    tabulated in {{{SubscriptBox[W,0,  0, 0],…}…}…}.";

SymmetricMeanW[n_Integer, wt_List] :=
 Apply[Plus, wt * HeterozygoteB[n], {2}]

SymmetricMeanW[i_Integer, j_Integer, n_Integer, wt_List] :=
    HeterozygoteB[i, j, n].wt[[i + 1, j + 1]];
```

```
SymmetricMeanW[i_Integer, j_Integer, n_Integer, W_?(Not[ListQ[#]] &)] :=
  Module[{l, hh = HeterozygoteB[i, j, n]},
            Sum[hh⟦i + j - 2 l + 1⟧ W[i, j, i + j - 2 l],
    {l, Max[0, i + j - n], Min[i, j]}]];

SymmetricMeanW[n_Integer, W_?(Not[ListQ[#]] &), opts___Rule] :=
 Module[{i, j}, Table[SymmetricMeanW[i, j, n, W], {i, 0, n}, {j, 0, n}]]
```

SymmetricNeutralHaploidFrequencies

```
SymmetricNeutralHaploidFrequencies::usage =
  "SymmetricNeutralHaploidFrequencies[{{ψ0,…},{u0,…}},n] represents
    a population by the frequency of each genotypic class,
    ψi, and the frequency of a neutral marker within that
    class, ui.  SymmetricNeutralHaploidFrequencies[{{ψ0,…},{u0,…}}]
    is shorthand for
    SymmetricNeutralHaploidFrequencies[{{ψ0,…},{u0,…}},1].";

SymmetricNeutralHaploidFrequencies[ψ_List] :=
  SymmetricNeutralHaploidFrequencies[ψ, 1];
```

SymmetricNeutralDiploidFrequencies

```
SymmetricNeutralDiploidFrequencies::usage =
  "SymmetricNeutralDiploidFrequencies[{{{SubscriptBox[ψ,0,
    0],…},…},{{SubscriptBox[u,0, 0],…},…}},n] represents a
    diploid population by the frequency of each genotypic
    class, SubscriptBox[ψ,i, j], and the frequency of a neutral
    marker within that class, SubscriptBox[u,i, j].
    SymmetricNeutralDiploidFrequencies[{{{SubscriptBox[ψ,0,
    0],…},…},{{SubscriptBox[u,0, 0],…},…}}] is shorthand for
    SymmetricNeutralDiploidFrequencies[{{{SubscriptBox[ψ,0,
    0],…},…},{{SubscriptBox[u,0, 0],…},…}},1].";

SymmetricNeutralDiploidFrequencies[ψ_List] :=
  SymmetricNeutralDiploidFrequencies[ψ, 1];
```

SymmetricRepresentations

```
SymmetricRepresentations::usage =
  "SymmetricRepresentations is a pattern which matches any
    symmetric representation.";

SymmetricRepresentations =
  SymmetricHaploidFrequencies | SymmetricDiploidFrequencies |
   SymmetricNeutralHaploidFrequencies |
   SymmetricNeutralDiploidFrequencies;
```

TruncationSelection

```
TruncationSelection::usage =
  "TruncationSelection->True is an option for SelectionExact
    and NewGametes which applies when individuals are selected.
    Those individuals with highest 'fitness', W, survive; if
    f[n,W̄] is larger than deme size, or if no f[] is specified,
    then all individuals survive.  If TruncationSelection->False
    (the default), then survival or reproduction occurs
    independently with probability W.";
```

APPENDIX II: R SCRIPT FOR SIMULATION AND EXCLUSION PROBABILITY ANALYSIS

```
## ## ## ## ## ## ## ## ## ## ## ## ## ##
#
# THIS SCRIPT ALLOWS YOU TO CREATE POPULATIONS IN R AND DEFINE EXCLUSION PROBABILITIES
#
## ## ## ## ## ## ## ## ## ## ## ## ## ##
```

```
rm(list=ls())

for (Numb_loci in 10:20) {
    for (Numb_alleles in 5:10) {
        for (M in 1:10) {

Numb_ind<-1000
Numb_Offspring<-100

x<-seq(from=100,to=(100+2*Numb_alleles-2),by=2)
# Having this will correspond to sampling di-nucleotide microsatellites,
# which is pretty common in nature.

x2<-seq(from=98,to=(100+2*Numb_alleles),by=2)
# This is just to get nicer histograms later

# Defining the allele frequency distribution #

ind<-c(1:Numb_alleles)
Bernatchez<-vector()
tot<-sum(ind/((Numb_alleles+1)-ind))
Bernatchez[ind]<-(ind/((Numb_alleles+1)-ind)*tot)

Distribut<-vector("list",3)

Distribut[[1]]<-rev(Bernatchez/sum(Bernatchez))
# Distribution by Bernatchez/Duchesne

Distribut[[2]]<-dgeom(0:(Numb_alleles-1),0.3)
# Geometric Distribution

Distribut[[3]]<-rep(1/Numb_alleles,Numb_alleles)
# Uniform Distribution

# Creating Adult genotypes from a specific allele frequency distribution #

Adult_genotypes<-vector()

for (i in 1:Numb_loci) {
Adult_genotypes<-cbind(Adult_genotypes,sample(x,2*Numb_ind,replace=T,
prob=Distribut[[1]]))
}

reproductive<-sample(seq(1,2*Numb_ind,by=2),Numb_Offspring*2)
# This allows us to define the index of individuals that will
# produce offspring
```

Since this is ramdom sampling, there's no selection going on (Just drift!). Moreover, since replacement = F by default in R, in these simulations we are allowing each adult to produce one descendant only. We could get a specific variance in family size by using a specific probability vector.

```
gametes<-vector("list",Numb_Offspring*2)

for (i in 1:(Numb_Offspring*2)) {

    nou<-rbind(Adult_genotypes[reproductive[i],],
    Adult_genotypes[reproductive[i]+1,])
    index<-sample(1:2,10,replace=T)
    # This is Binomial sampling of gametes (i.e. Linkage Equilibrium)

        for (j in 1:Numb_loci) {
            gametes[[i]][j]<-nou[index[j],j]
            }
    }
```

```
index_gamet<-sample(1:200)
True_offspring_genotypes<-vector()
False_offspring_genotypes<-vector()

# True Offspring comes from the Adult population #

for (i in 1:(Numb_Offspring*2)) {
True_offspring_genotypes<-rbind(True_offspring_genotypes,
gametes[[index_gamet[i]]])
}

# False Offspring comes from a new population,
# now with a different allele frequency distribution  #

for (i in 1:Numb_loci) {
False_offspring_genotypes<-cbind(False_offspring_genotypes,
sample(x,2*(Numb_ind-Numb_Offspring),replace=T,prob=Distribut[[3]]))
}

Offspring<-rbind(True_offspring_genotypes,False_offspring_genotypes)
# Like this we place both true and false offspring into one Table
```

With the following bit of code we create a 2-column genotype table for both parents and adults

```
mig<-vector()
mig2<-vector()
Offspring_1col<-vector()
Adult_genotypes_1col<-vector()

for (j in seq(1,2*Numb_ind,by=2)) {
        mig<-Offspring[j,]
        mig2<-Adult_genotypes[j,]

        for (k in 1:Numb_loci) {
            mig<-append(mig,Offspring[j+1,k],2*k-1)
            mig2<-append(mig2,Adult_genotypes[j+1,k],2*k-1)
            }

    Offspring_1col<-rbind(Offspring_1col,mig)
    rownames(Offspring_1col)<-paste("Offspring",
    seq(1,length(Offspring_1col[,1])),sep="-")
    colnames(Offspring_1col)<-paste("Locus",
    rep(seq(1,Numb_loci),each=2),sep="-")

    Adult_genotypes_1col<-rbind(Adult_genotypes_1col,mig2)
    rownames(Adult_genotypes_1col)<-paste("Adult",
    seq(1,length(Adult_genotypes_1col[,1])),sep="-")
    colnames(Adult_genotypes_1col)<-paste("Locus",
    rep(seq(1,Numb_loci),each=2),sep="-")

}

resum<-paste("Offspring_1col-",Numb_loci,"-loci-",
Numb_alleles,"-alleles-run-",M,".txt",sep="")
write.table(Offspring_1col, file=resum,sep="\t")

resum<-paste("Adult_genotypes_1col-",Numb_loci,"-loci-",
Numb_alleles,"-alleles-run-",M,".txt",sep="")
write.table(Adult_genotypes_1col, file=resum,sep="\t")

Adults=Adult_genotypes_1col
Offspring=Offspring_1col
```

Now we begin with the calcualtion of Pr(Z) and Pr(delta)

```
massive=function(massive){

    AAT40=c(Adults[,a],Adults[,(a+1)])
    locus=as.data.frame(table(AAT40))
    Frequency=locus[,2]/sum(locus[,2])
    Data=cbind(locus, Frequency)
# Thus, we have a dataframe with 3 columns that includes:
# Alleles/Abs.freq/Rel.freq
    if (Data[1,1]==0) {Data=Data[-1,]} else Data=Data
```

```
# In case there's null allele = 0!
    n1=((length(AAT40))/2)
# Total number of individuals sampled - assuming diploidy!

    A1=(Data[,3]*(2*n1))
    A2=(Data[,3]^2)*n1
    AA=A1-A2
    AAA=cbind(Data,AA)
# Thus, we're adding a column that includes the 2z-(z)2

# Now we do exactly the same for the offspring!

    AAT402=c(Offspring[,a],Offspring[,(a+1)])
    locus2=as.data.frame(table(AAT402))
    Frequency2=locus2[,2]/sum(locus2[,2])
    Data2=cbind(locus2, Frequency2)
    if (Data2[1,1]==0) {Data2=Data2[-1,]} else Data2=Data2
    n2=((length(AAT402))/2)

    B1=(Data2[,3]*(2*n2))
    B2=(Data2[,3]^2)*n2
    BB=B1-B2
    BBB=cbind(Data2,BB)

# Now we focus on the alleles that are found both in the parentals
# and in the offspring.
# It is important to note that alleles occurring in only one sample
# (i.e. adults or juveniles) will not be included in the calculation
# because the product equals zero.

    AAA1=match(BBB[,1],AAA[,1])
    g=which(AAA1>0)
    g1=AAA1[g]
    AAA1=AAA[g1,]
    BBB1=match(AAA[,1],BBB[,1])
    j=which(BBB1>0)
    j1=BBB1[j]
    BBB1=BBB[j1,]

# So we take the product and sum it

    AB=AAA1[,4]*BBB1[,4]
    AB=sum(AB)

# Now we make two columns with the different allele combinations
# and get rid of the 'homozygotes' using the line:
# which(Agenotypes[,1]==Agenotypes[,2]).
# Please, notice that now we are dealing with the shared alleles
# from parents (AAA1) and offspring (BB1)

    y=AAA1[,1]
```

```
    Agenotypes=expand.grid(y,y)
    remove2=which(Agenotypes[,1]==Agenotypes[,2])
    Agenotypes=Agenotypes[-remove2,]

    z=BBB1[,1]
    Bgenotypes=expand.grid(z,z)
    remove2=which(Bgenotypes[,1]==Bgenotypes[,2])
    Bgenotypes=Bgenotypes[-remove2,]

# This allows us to calculate the genotype frequencies in Adults
# and Offspring

    one=match(Agenotypes[,1],AAA1[,1])
    two=match(Agenotypes[,2],AAA1[,1])
    one=AAA1[one,3]
# Like this we get a vector that includes allele frequencies instead
# of allele names
    two=AAA1[two,3]
# Like this we get a vector that includes allele frequencies instead
# of allele names, but for the second column
    Agfreq=one*two*n1*2

    oneb=match(Bgenotypes[,1],BBB1[,1])
    twob=match(Bgenotypes[,2],BBB1[,1])
    oneb=BBB1[oneb,3]
    twob=BBB1[twob,3]
    Ogfreq=oneb*twob*n2*2

    Gfreqs=Agfreq*Ogfreq
    Gfreqs=floor(Gfreqs)
    Gfreq=sum(Gfreqs)/2

    PrB=(AB-Gfreq)/(n1*n2)
# So this is the probability of a randomly selected dyad
# from a particular locus sharing an allele

    nom<-paste("PrZ-",Numb_loci,"-loci-",
    Numb_alleles,"-alleles-run-",M,".txt",sep="")
    write.table(PrB,file=nom,row.names=FALSE,col.names=F,
    sep="\t",append=T)

    }

al=ncol(Adults)

C1=for(a in seq(1,al,2)) lapply(a,massive)

nom<-paste("PrZ-",Numb_loci,"-loci-",
    Numb_alleles,"-alleles-run-",M,".txt",sep="")
PrBs <- read.table(nom, header=F, sep="\t",
    na.strings="NA", dec=".", strip.white=TRUE)
```

```
nn1=length(Adults[,1])
nn2=length(Offspring[,1])

Pr_delta = prod(PrBs[,1])

Expected_Number_of_False_Pairs = Pr_delta*(nn1*nn2)

pvalue1=cbind(Expected_Number_of_False_Pairs,Pr_delta)

nom2<-paste("Fpairs+Prdelta-",Numb_loci,"-loci-",
    Numb_alleles,"-alleles.txt",sep="")
write.table(pvalue1,file=nom2,row.names=FALSE,sep="\t",append=T)
```

Begin dyad sorting. Note that allele sizes must be the same order of magnitude

```
Anames=rownames(Adult_genotypes_1col)
Onames=rownames(Offspring_1col)

categories=ncol(Adults)
Aindivids=length(Adults[,1])
Oindivids=length(Offspring[,1])

A=1:Aindivids
O=1:Oindivids
G=expand.grid(A,O)
AG=G[,1]
AO=G[,2]
Ads=Adults[AG,]
Offs=Offspring[AO,]

IdnamesA=Anames[AG]
IdnamesO=Onames[AO]
write.table(IdnamesA,file="IdnamesA.txt",
    row.names=FALSE,col.names=F,sep="\t",append=F)
write.table(IdnamesO,file="IdnamesO.txt",
    row.names=FALSE,col.names=F,sep="\t",append=F)
IdnamesA<-read.table("IdnamesA.txt",header=F,
    sep="\t",na.strings="NA",dec=".",strip.white=TRUE)
IdnamesO<-read.table("IdnamesO.txt",header=F,
    sep="\t",na.strings="NA",dec=".",strip.white=TRUE)
Names=cbind(IdnamesA,IdnamesO)

matches=function(matches) {
A=Ads[,z]-Offs[,z]
B=Ads[,(z+1)]-Offs[,(z+1)]
C=Ads[,z]-Offs[,(z+1)]
D=Ads[,(z+1)]-Offs[,z]
f=A*B*C*D
f=f^2
f=cbind(z,f)
```

```
write.table(f,file="Sort.txt",row.names=FALSE,
col.names=F,sep="\t",append=T)}

z=ncol(Ads)
C1=for(z in (2*(unique(round((1:(z-2))/2)))+1)) lapply(z,matches)

Observed<-read.table("Sort.txt",header=F,sep="\t",
    na.strings="NA",dec=".",strip.white=TRUE)
a=unique(Observed[,1])
U=NULL

for (i in a) {u=Observed[Observed[,1]==i,2] U=cbind(U,u)}

a=length(U[,1])

stuff=rowSums(U)

Sorted=cbind(Names,stuff)
matches=which(Sorted[,3]==0)

Actual=sort(stuff)
IDS=which(stuff==0)
PAdults=Ads[IDS,]
POffspring=Offs[IDS,]

nput=length(matches)
Putativepairs=Sorted[matches,]
Phi=Expected_Number_of_False_Pairs/nput
if (Phi>1) {Phi=1}
Phi=cbind("Phi",Phi)

nom3<-paste("Phi-",Numb_loci,"-loci-run",M,".txt",sep="")
write.table(Phi,file=nom3,row.names=FALSE,col.names=F,
sep="\t",append=T)

unlink("Sort.txt")
unlink("IdnamesA.txt")
unlink("IdnamesO.txt")

}
}
}
```

This will end the exclusion probability calculations

Using a *Mathematica* notebook

Mathematica has a front end and a kernel, which work independently: one handles formatting, the other the calculations. If the kernel crashes, the calculations are lost; if the front end crashes, the current notebook is lost. Save regularly!

To find out about a function, use ?name, and then type [enter]. For example:

```
? IterateCline
```

```
Information::notfound : Symbol IterateCline not found. ≫
```

This also works for inbuilt functions. You can find out more about them using the Help... menu; packages are documented under Add-Ons.

Cells are structured by headings, subheadings etc. Double click on the right-hand brackets to open & close them.

To find out where do you have to install your add-ons or where is your Base Directory you type:

```
$UserBaseDirectory (*This is where you have your Mathematica packages*)
```

```
C:\Documents and Settings\Phyllamphion\Datos de programa\Mathematica
```

To find out where does *Mathematica* send the files containing the genotypes from simulated individuals you type:

```
Directory[] (*This is where the files are going to*)
```

```
C:\Documents and Settings\Phyllamphion\Mis documentos
```