AN ABSTRACT OF THE THESIS OF

Bruce A. Peterson for the degree of Doctor of Philosophy in Statistics presented on December 15, 1986 .

Title: Resource Limited Competition of Two Species: A Dynamic Model of a Perturbed Neutrally Stable System

Redacted for privacy

Abstract Approved: W. Scott Overton

An hierarchically structured resource flow competition model for two species is developed. The model is shown to be intrinsically neutrally stable with a dynamical behavior derived from perturbation responses. The model properties are contrasted with those of a conventionally interpreted Lotka-Volterra competition model.

A simulation developed from the model is compared with the experimental results of Gause's 1932 competition experiment. The simulation reproduces the dynamical behavior of the experimental population and provides additional insight into potential alternative behaviors not obtained from the conventional Lotka-Volterra approach.

The resource flow competition model offers an alternative viewpoint in understanding the behavior of competing species. The model development methodology has the advantage that parameters are tied to physical processes at a known hierarchical resolution and the dynamics of the perturbed neutrally stable system offer a rich repertoire of behavior. A model is developed in which the focus is on the extrinsic rather than in intrinsic aspects of system behavior.

Resource Limited Competition of Two Species: A Dynamic Model of a Perturbed Neutrally Stable System

By

Bruce Peterson

A THESIS

submitted to

Oregon State University

in partial fulfillment of
the requirements for the
degree of

Doctor of Philosophy

Completed December 15, 1986

Commencement June 1987

APPROVED:

Redacted for privacy

Professor of Statistics in charge of major

Redacted for privacy

Head of department of Statistics

Redacted for privacy

Dean of Graduate School

Thesis presented on December 15,1986

Prepared by Bruce Peterson

Table of Contents

List of Figures

Resource Limited Competition of Two Species: A Dynamic Model of a Perturbed Neutrally Stable System

Section I
Introduction

Three themes are intertwined in the following development. A tool, hierarchical systems theory, is applied to a concept, resource flows as a primary ecological process, to explore a basic ecological process, species competition. Each of these will be discussed briefly in the introduction before the full details of model are developed.

Energy flow as a primary ecological process was hypothesized by Odum (1976) in terms of the maximum power principle. According to this principle, systems will tend to maximize the energy flow through their domain when energy is in surplus, and will maximize their efficiency when the surplus is no longer available.

The hypothesis provides an organizational principle for a holistic perspective on the behavior of complex systems, providing an expected behavior for systems at different levels of organization and a basis for organizational structure. Hypothesized systems, especially at the supra-organismal level, sometimes lack otherwise obvious

behaviors. The principle provides an immediate set of hypotheses which can be used to explore the nature of an ecological system.

The exploration the organizational behavior of an hypothetical system following this principle requires the explicit inclusion of energy and resource flows. Such a model will provide testable hypotheses more readily than models in which energy or resource flows are included implicitly.

One aspect of the principle, the maximization of energy flow through a system, immediately suggests the class of model which would be advantageous. The attempt by several systems, at the same level of organization, to maximize energy flow suggests behavior analogous to competition for a limited resource. Maximization of an energy flow into a system is advantageous simply because energy used by a system is not available for use by the system's competitors.

Models of competitive behavior have been central to much of ecological theory. Competition models have been used in

discussing aspects of ecological systems ranging from community structure (Whittaker,1975), trophic structure (Pimm, 1982), and evolutionary paths (Pianka,1974). Pianka summarized the focus on competition models in stating "Natural selection and competition are inevitable outgrowths of reproduction in a finite environment."

That a concept and its associated models are applicable to such a wide range of biological systems and behaviors is also an indication that the model elements may have differing interpretations. A language that allows models of complex systems to be developed with a minimum of confusion is required.

Such a language is expressed in the theory of hierarchical systems. The basic elements of the theory consist of three concepts: levels of organization, statistical closure, and near decomposability.

Multiple levels of organization in models of complex systems stem from the recognition that complex systems have behavior occurring simultaneously at several time

resolutions (Simon, 1973). Models simulate this by consisting of modules for each class of characteristic times. Lower levels are defined as consisting of subsystems with shorter characteristic times while subsystems with longer characteristic times represent higher levels.

One outcome of the division into levels is that subsystems which are at different levels will interact differently than subsystems which are at the same level (Simon,ibid.). Lower level systems will interact with higher level systems through their aggregate behavior. Higher level systems will act on systems at a lower level by parametric coupling. That is, the parameters of the lower level will be functions of the higher level behavior. Pattee (1973) described this property as statistical closure, considered a necessary element for hierarchical control systems.

Near decomposability, the last element from the hierarchical theory, summarizes the principle that interactions between subsystems will be fewer and weaker than the interactions within subsystems. Simon (ibid.), observes that near-decomposability of subsystems contributes to the robustness of the whole system through

the damping of the propagation of disruptions caused by failures of any one subsystem.

The concepts used in the development of the resource competition model are developed in the following sections.

The role of competition in ecological theory is discussed in the second section. The competition of species for resources is a central concept in ecological theory, based primarily on a development by Lotka (1925) and Volterra (1926) as a differential equation model. This model has provided the basis for a large number of ecosystem models.

The third section explores the structural elements of a Lotka-Volterra competition model. The form of this model is simplified by transformation into resource normalized units which explicitly model competition for a single resource flow.

The equilibrium dynamics of the resource competition model is simple. However the behavior is rich when the system is stochastically perturbed. Since the time scales of dynamics at different levels are necessarily slower for higher

levels and faster for lower levels than the characteristic time of the target level, such a coupling may be modeled as a series of perturbations to the target level. Section 4 explores analytically the behavior of the competition model with a simple perturbation.

A model is developed for the perturbations in terms of predation in section 5. This model differs in important respects from classical predator-prey models. The resource competition model explicitly considers the 'predation' events to be infrequent compared to the characteristic times of the competing populations. In contrast, the usual predator-prey or community model considers the characteristic times of all species, predators and prey, to be similar.

The predation model generates perturbations in the state variables of the target level model. Section 6 investigates the behavior of the target model when the predation perturbations range from density independent to density dependent. Resource perturbations model are discussed in section 7.

Section 8 extends the model to populations of the competitive systems. The integration of system behavior produces the next hierarchical level and allows the distribution of competition systems states to be explored.

Section 9 compares the theoretical results obtained from the model to Gause's classic experimental results using the parameters estimated by Gause (1934).

Section 10 summarizes the investigation. The general properties of the model and crucial aspects are discussed and reviewed.

Section 2

Competition and Ecosystem Models

The concept of competition for limited resources has played a central role in the development of ecological theory since Darwin (1859). Recent literature includes discussions of aspects of ecological systems ranging from community structure (Whittaker, 1975), trophic structure (Pimm,1982), evolutionary paths (Pianka, 1974) and natural selection (Fisher, 1930) which rely on the concept of competition as the core of the patterns developed.

Lotka (1925) and Volterra (1926) provided the differential equations that are the starting point of many competition models. This model of two species competition can be expressed as:

(1) $\dot{N}_X = r_X N_X(1-N_X/K_X - \beta N_Y/K_X)$

$\dot{N}_Y = r_Y N_Y(1-N_Y/K_Y - \gamma\beta N_X/K_Y)$

where

r_X is the per capita growth rate for species X, at low values of N_X, the intrinsic growth rate

r_Y is the per capita growth rate for species Y, at low values of N_Y, the intrinsic growth rate

K_X is the carrying capacity for species X with no competition

K_Y is the carrying capacity for species Y with no competition

β is the number of species X equivalent to an individual of species Y

γ is the number of species Y equivalent to an individual of species X

These equations contain a number of implicit assumptions about the populations being modeled.

Species populations are assumed to increase exponentially until limited by intraspecific or interspecific competition. The nature of the competition is not specified in the model but there are many ways in which two species can compete. Schoener (1983), for example, lists seven mechanisms of competition:

> Consumptive (or resource) competition in which one individual by consuming a resource denies it to all other individuals.
>
> Preemptive competition in which a unit of resource is "occupied" by an individual denying the resource to others
>
> Overgrowth, in which one or more individuals grow over or upon another and either damage it or deny it access to a required resource

Chemical competition in which an individual releases chemicals into the environment which inhibit or harm competing individuals

Territorial competition in which an individual defends a unit of space by aggressive behavior against competitors.

Encounter competition, in which interactions between mobile organisms result in harm to one or more of them.

These mechanisms can be summarized in two categories: resource or exploitative competition, and interference competition. A species engaged in resource competition denies resources to a competitor by first use. A species engaged in interference competition denies resources to a competitor without using the resources. The first of Schoener's mechanisms is clearly resource competition. The second has elements of both resource competition and interference competition, but its effect is resource competition even though the mechanism is interference. The rest are interference competition.

Further assumptions are implicit in the differential equation models:

- the population members are nearly identical; no differences due to age, sex, size, etc.

- the population growth rates respond instantaneously to changes in resources or competition.
- the mechanism of competition is unspecified and proportional to total population density.

The characteristic time of the model, though unspecified, must be sufficient for the other assumptions to be approximately correct. Often this implies that more complex systems i.e. the higher the hierarchical level that is being modeled, have the longer characteristic times.

The unspecified mechanisms of competition are expressed through the parameters beta and gamma. These parameters limit the maximum population numbers to a value of K; resources are not explicitly modeled.

Field studies, purporting to identify competition effects, and therefore indirectly the model parameters, have had a variety of interpretations and success rates. For example, Connell (1983) argues from a review of 72 field studies published from 1972 through 1982 that interspecific competition was demonstrated in only about half of the multispecies systems studied. On the other hand Schoener

(1983) surveyed 164 field studies published through 1982 and concluded that 90% of the experimental attempts to detect interspecific competition did so.

The differences in interpretation may stem in part from the different models of competition used. Any attempt to measure an effect is dependent on the model of how the effect will be expressed. Since much of competition theory has been built on a particular model of competitive interaction, some of the disagreements about experimental results may be the result of differing interpretations of this model.

Variants of the competition model have been developed to accentuate one or another aspect of the species interaction. A common approach is to regard the model as but the first few terms of a series approximation to a more complex model. This mathematical approach has been used since the work by Lotka (1925) to create more generalized models of competition.

For example a model used by Tilman (1982) and a similar model by MacArthur (1972) are such models. The model has

been expanded to include terms that explicitly characterize resources with a general form of:

(2) $\dot{N}_i = N_i(f_i(R_1 \ldots R_k) - m_i)$

$\dot{R}_j = g_j(R_j) - \Sigma^n_{i=1} N_i f_i(R_1 \ldots R_k) h_{ij}(R_1 \ldots R_k)$

where there are $i=1 \ldots n$ species with population density N_i

and $j=1 \ldots k$ resources with availability R_j

the m_i are the per capita mortality rates for species i

the g_j describe the rate of resource production

the f_i describe the net per capita reproductive rate depending on resource availability

the h_{ij} describe the amount of resource j needed for each new individual of species i

Resources are treated as state variables in this model. The characteristic time of the resource dynamics is commensurate with the characteristic times of the populations. In contrast recall that in the model of equation 1, resources appear implicitly as parameters; implying that the characteristic time of resource dynamics is long compared to that of population dynamics. The implicit assumptions of both types of model are otherwise similar. That is, populations are assumed to

instantaneously respond to any change in resource availability, each member of the population is indistinguishable from any other, and all members interact only through the use of resources.

The implicit resource model assumes that the resources are available as a "flow" only. Those resources not used by the populations do not accumulate and resources depleted at one time do not affect availability at a later time. The implicit resource model puts resource dynamics on a different, higher, level than the population dynamics. In contrast the state variable model (model 2) assumes that resources and populations are at the same hierarchical level; the characteristic times of populations and resources are sufficiently close that the joint dynamics are of interest.

As a result, the two models of competition for resources are fundamentally different. In particular, two species competing for a resource flow will not experience a time lag in resource availability. A quantity of resource relinquished by one population is immediately available to

its competitors; there is no lag while resources recover from a depressed level. Some authors have recognized this: for example Smith (1952) early discussed the importance of resource flows while Odum (1976) has developed a suite of resource flow models and a modeling language to describe them.

The state variable type model is often implicitly or even explicitly invoked when the parametric model is analyzed. That is, the resource is often conceived of and discussed as a spectrum or gradient of resources (e.g. seed sizes) at the same hierarchical level. This is the case in the analyses by MacArthur, Pianka, May (previously cited) and Diamond (1975).

Section 3

A Resource Competition Model

An implicit resource model can be developed which, in contrast to the mathematical expansion approach, recognizes that the parameters of the model are the result of higher level constraints. As such, the model can be reformulated to make explicit the competition for resources as a constraint from a higher level of organization.

A benefit of reformulating the model in terms of resources is that the biological meaning of the parameters is made clearer and the values available to certain parameters are constrained. This aids in the comprehension of the modeled system and in the identification of the limiting factors as promoted by Smith (1952).

The reformulation of equation 1 to a resource basis begins by recalling that at equilibrium the solution to this set of differential equations is:

(3) $$N_X^* = (K_X - \beta K_Y)/(1-\gamma\beta)$$

$$N_Y^* = (K_Y - \gamma K_X)/(1-\gamma\beta)$$

The solution is in terms of the population densities and

carrying capacities. The equations (1) can be reformulated to make the parameters resource related by the transformation:

let $K_X = R/\Phi_X$

$K_Y = R/\Phi_Y$

and

$\beta = \delta_Y\Phi_Y/\Phi_X$

$\gamma = \delta_X\Phi_X/\Phi_Y$

where

Φ_X is the per capita resource use by species X

Φ_Y is the per capita resource use by species Y

and

δ_Y is the resource preemption factor of species Y on species X

δ_X is the resource preemption factor of species X on species Y

Note: the preemption factor is the ratio of resources a species denies its competitor compared to the relative per capita resource usage of the two species

then

$$\dot{N}_X = r_X N_X(R - \Phi_X N_X - \delta_Y\Phi_Y N_Y)/R \quad (4)$$

$$\dot{N}_y = r_y N_y (R - \Phi_y N_y - \delta_x \Phi_x N_x)/R$$

and the equilibrium solution is:

$$N_x^* = R(1-\delta_y)/\ \Phi_x(1-\delta_x\delta_y)$$

$$N_y^* = R(1-\delta_x)/\ \Phi_y(1-\delta_x\delta_y)$$

The two pairs of equations, the differential form and the equilibrium solution form, define a phase space and isoclines in that phase space. Figure 1 illustrates the phase space created with the population density of one species on the ordinate and the population density of the other on the abscissa. The isocline consists of the locus of points in the phase space where the time derivative of the population densities of each species is zero.

When the preemption factors (delta) are not one, an isocline is defined for each differential equation. The intersection of these isoclines is a node in the phase space. The classical analysis of this node (see Rosen, 1970) shows that the node is stable for both delta's less than one and asymptotically unstable for both delta's greater than one.

When the delta's are identically one, the equilibrium

solutions obtained in equations 4 are indeterminate. Rather than a unique equilibrium point in the phase space, there exists a locus of equilibrium solutions on a line. This equilibrium line is illustrated in figure 1.

This formulation of the model of equation 1 has all the parameters of the model in common resource terms. The biological meaning of the each term of the reformulated model is now clearer since the intra and inter species interactions are in terms of the per capita resource usage and a resource preemption factor.

The resource used by both species is represented by one term, R. This is consistent with the "law of the minimum ". That is, at any given time, only one resource will be the limiting resource. The specific resource which is limiting may shift from time to time and the effect on the dynamics of the system will depend on the nature of the shift.

Figure 1

State Space of Two Species Competition Model

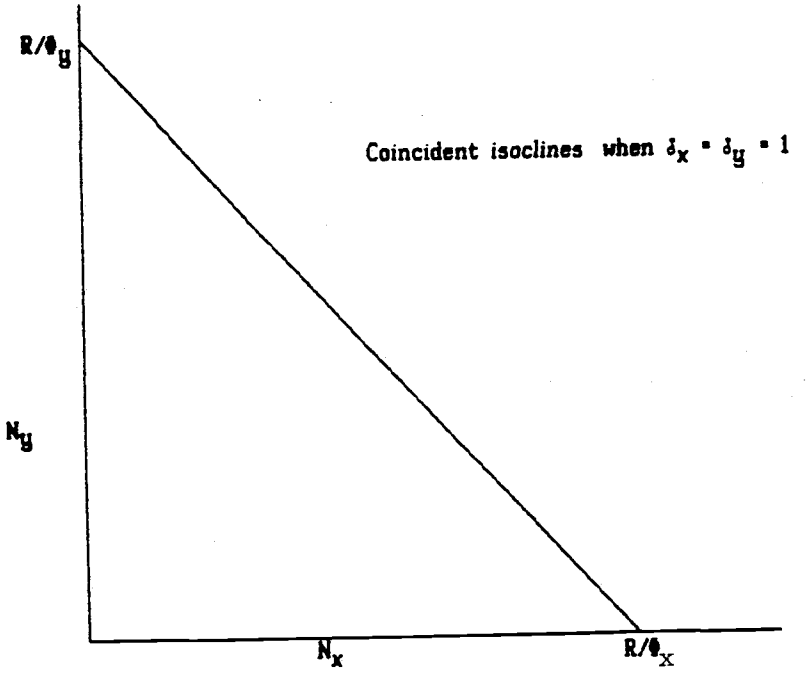


Figure 1: Phase Space of Two Species Competion Model
The isoclines for the two species are coincident when the δ_X and δ_Y are both 1. Representation of each species in "resource equivalent units", $N_X'=\Phi_X N_X$ and $N_Y'=\Phi_Y N_Y$, leads to a simplified form. This results in a figure as above with $\Phi_X = \Phi_Y = 1$.

If a second resource becomes limiting for the same two species, then this is equivalent to a new isocline sweeping through the phase plane. When this new isocline passes through the current system state, it will pick up the equilibrium state and carry the state on the new limiting isocline.

The absolute population numbers may change but the population numbers may be renormalized. The renormalization is accomplished by absorbing the efficiency parameters, phi, into the population numbers to create new variable measured in resource demand equivalents. Since the efficiency parameters only affect the slope of the isocline, this transformation does not change the dynamics of the system and produces an isocline which can be analysed with a slope of minus one.

Model 4 can be renormalized by incorporating the parameters phi into the state variables so that the population values, N_X and N_Y are expressed in terms of resource demand equivalents rather than numbers of individuals. This renormalization is expressed in the following form in the discussion that follows.

(5) $\dot{N}_x = r_x N_x (R - N_x - N_y)/R$

$\dot{N}_y = r_y N_y (R - N_x - N_y)/R$

N^*_x, N^*_y are not uniquely defined

If a second resource becomes limiting for one species, but is not limiting for the second, then competition with either species may occur with a third, new, species. In this case the new equilibrium value for the original species will be unrelated to the old equilibrium value, even if renormalized. Thus the reaction of the competitive system to a change in resource composition depends on the community structure in which it is embedded.

The parameters of interaction, delta, include all interactions through resource competition. The following model will explore the interaction by assuming that species are competing for two different resources with some overlap and then examining what occurs when the overlap becomes complete; i.e. the species are competing for one resource. This will provide an explicit definition of the parameters delta in terms of resources.

Suppose that each species has separate resources available so that the model is:

$$(6) \qquad \dot{N}_X = r_X N_X(1 - (N_X/R_X) - (\delta_Y N_Y/R_X)$$

$$\dot{N}_Y = r_Y N_Y(1 - (N_Y/R_Y) - (\delta_X N_X/R_Y)$$

let R be the total resources available and R_C be resources for which both species compete, then

$$R = R_X + R_Y - R_C$$

Now the δ's can be defined as:

δ_X = the proportion of R_X available to species Y

$$= R_C/R_X$$

δ_Y = the proportion of R_Y available to species X

$$= R_C/R_Y$$

Note that if $R_C = 0$ then there is no competition and $\delta_X = \delta_Y = 0$.

If $R_C > 0$ but $R_C < R_X$ and $R_C < R_Y$ then each species has resources unavailable to the other; then $\delta_X > 0$ and $\delta_Y > 0$ but both are less than 1. Then the equilibrium populations

$$N_X^* = (R_X - \delta_Y R_Y)/(1 - \delta_X \delta_Y)$$

$$N_Y^* = (R_Y - \delta_X R_X)/(1 - \delta_X \delta_Y)$$

are stable. Only if $R_C < R$ and $R_C = R_X$ or $R_C = R_Y$

does one species exclude the other; each species persists if it has resources unavailable to the other.

The last case explores the model of interest in this investigation. In this case $R_C=R_X=R_Y=R$, both species are competing for the same resource, and this is the condition considered here as resource competition. There is no unique equilibrium in this case as $\delta_X=\delta_Y=1$. The population system is neutrally stable.

Thus, the interaction parameters of a resource competition model are 1.0 and this produces a dynamical system which is neutrally stable. That is, any perturbation of the system will tend to change the system state permanently. Ordinarily this would be regarded as a structurally unstable system; the structural instability results from the precise equality of the parameters needed to maintain neutral stability in the phase space. However in competition for one resource, the interaction parameters are 1 by definition.

The fact that a neutrally stable system is ordinarily considered structurally unstable and therefore presumably a

special uninteresting case is reflected in much of the literature discussing the competition relations. Generally the neutrally stable case is not examined in any detail. For example Slobodkin (1961) and Pianka (1974) who both develop the competition system dynamics in terms very similar to those used here, leave out the neutrally stable case altogether. This case, however, appears in the present development as the most interesting of all for describing the behavior of the resource limited competitive system.

Further the neutral stability of the two-species, one-resource model is convenient for examining hierarchical interactions. Interactions with higher levels will take place through the parameters of the model. There are three types of these parameters: the growth rates, the resource efficiencies, and total resource.

The growth rate parameters (r_x and r_y) of the model are modified by the density dependent terms to yield the growth rates. A change in these parameters is structurally similar to the changes already occurring from the density dependent term and does not produce any new

behavior. However it does change the relative behavior of the species and the partition of the phase plane.

The resource efficiency parameters set the scale of the populations. Any change in these is equivalent to a renormalization; to changing the measure of population size.

A change in the total resource is the result of dynamics at a different hierarchical level. The effect of changes in resources, will be investigated in a subsequent section.

The population numbers, being the state variables of the model, are the result of the integrated behavior of lower level dynamics. This integrated behavior is smooth as long as the lower level dynamics are continuous. Should the lower level dynamics become discontinuous, the discontinuity would be reflected as a perturbation in the population numbers. This can generate new behavior which will be discussed in the next section.

Section 4

Perturbation Behavior

Model 5 developed in the previous sections has state variable dynamics at the population level which exhibit neutral stability. These state variables are the result of the integrated behavior of an unspecified lower level dynamics. A discontinuity in the lower level dynamics will be modeled as a perturbation to the target population level dynamics. The behavior of this system can differ markedly when perturbed in different ways. This section will develop an analytical approach to a simple type of perturbation.

Consider a model of perturbation in which the system state is moved from equilibrium on the isocline to a point on a semicircle centered on the original state. In this model the size of the perturbation in numbers of resource equivalent units will be constant, but the proportion of membership change for each species will be a random variable. Biologically, this type of perturbation will arise if the competing species experience perturbations (population losses or gains) independent of population size.

Although this density independent type of perturbation is simple, it can generate complex behavior in the neutrally stable two species resource competition system. In contrast, an asymptotically stable or unstable system has dynamics which dominate small perturbations of this type. However in a neutrally stable system, any movement of the system state upon recovery from a perturbation will be retained until the next perturbation. Thus if the perturbation is random then the change in state of the system will also be random with the distribution of the state change determined by the system dynamics and the distribution of the perturbations.

The distribution of the change in system state after random perturbation can be examined analytically for the simple perturbation presented above. Figure 2 presents the notation which will be used here. Angles used are in reference to the line parallel to the horizontal axis and intersecting the isocline at the original equilibrium state of the system.

The size of the perturbation is held constant in this analysis. The perturbation can then be treated as a vector where the angle that the vector makes with the reference line is theta. Theta is a random variable.

After perturbation, the system is in a new, non-equilibrium state. The system dynamics will return the system to equilibrium along a trajectory which depends the parameters of the dynamical relations. For small perturbations, this trajectory can be approximated by a line tangent to the trajectory at the perturbed system state. Figure 3 presents the accuracy of this approximation over a range of theta. For perturbations of one percent of population size, the approximation is reasonably good; the maximum error is also about one percent. The expected error depends on the position of the pre-perturbation state on the isocline as well as the parameter r_x and r_y. Since the error is systematic, there can be patterns in the behavior generated by this error.

Figure Two

Perturbation Notation

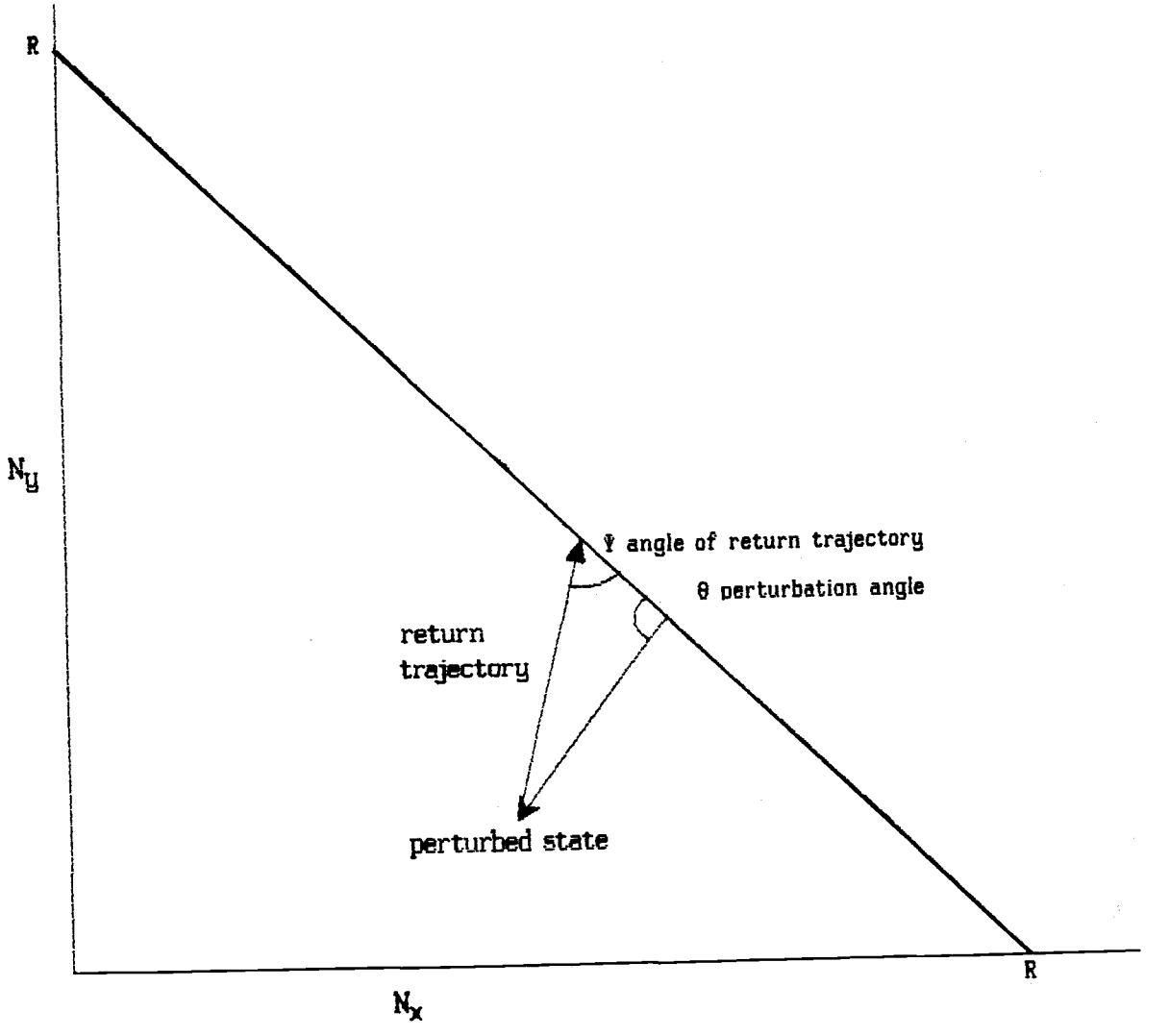
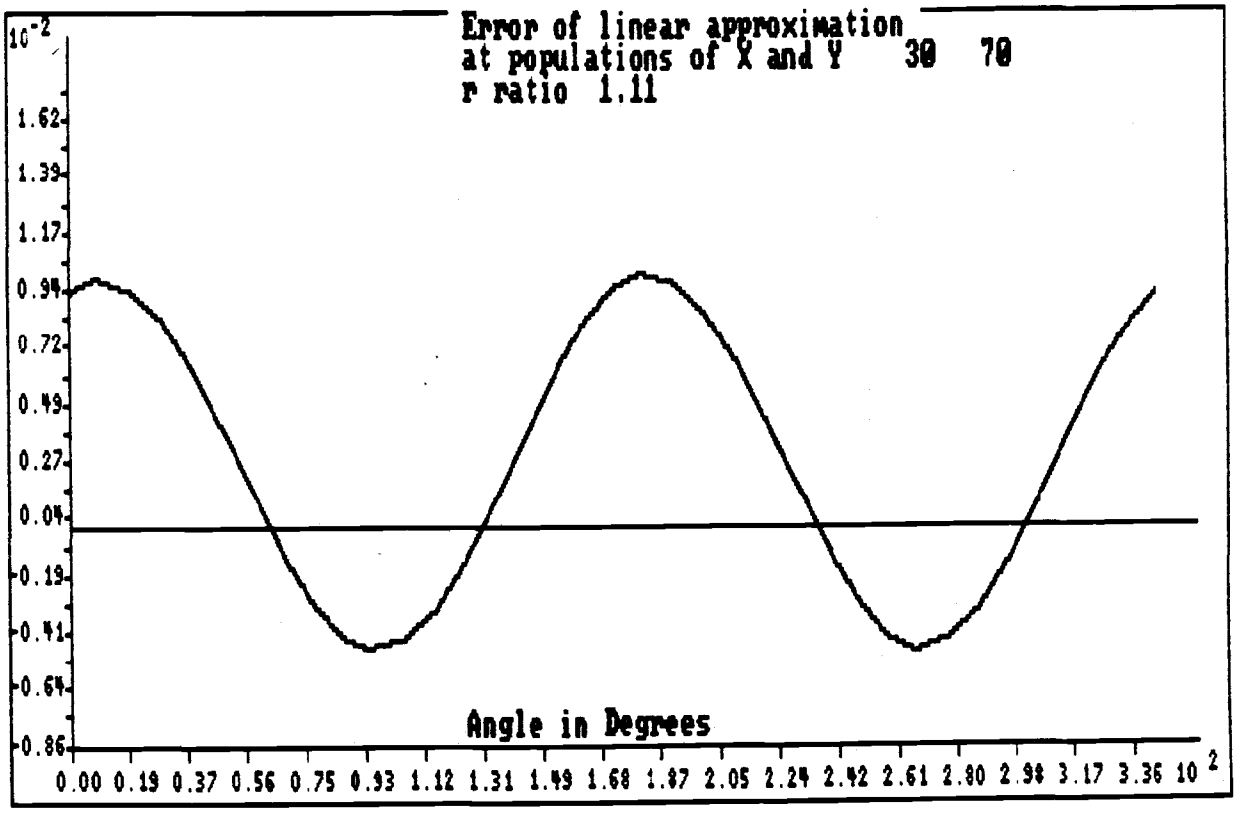


Figure 2: Notation used for perturbation of neutrally stable system. Note that perturbation size is exagerated.

Figure 3

Perturbation Recovery Approximation



In figure 2, Ψ is the angle of the tangent to the return trajectory and the isocline. The efficiency coefficients, Φ were incorporated into the population numbers in section 3 so that the N's now refer to resource equivalent population units. In this notation the angle of the isocline with the abscissa is then a constant of value π. Psi is then given by:

$$(7)\quad \Psi = \arctan(r_X N_X + r_Y N_Y)/(r_X N_X - r_Y N_Y))$$

The angle of the perturbation vector with respect to the abscissa is given by theta. For this development, theta is taken to be a random variable while the length of the perturbation vector, Z, is taken to be constant. Theta is additionally assumed to be drawn from a Uniform$(0,\pi)$ distribution.

Given that theta is then drawn from a uniform distribution with a range from 0 to π, the task is to find the distribution of d, where d is the signed distance from the original system state on the isocline to the final resultant state on the isocline after perturbation. The

signed distance and the distribution of the distance is given by:

(8) $d = z \sin(\Psi + \theta)/ \sin \Psi$

The cumulative frequency distribution is :

$F(d) = \int_0^{\theta'} d\theta/\pi$

where $\theta' = \arcsin(d \sin\Psi/z) - \Psi$.

This must be integrated piecewise over the four regions of d (where two of these overlap).

For $z < d <= z/\sin\Psi$ (z out to $z/\sin\Psi$):

$F(d)=\theta'/\pi$

For $z/\sin\Psi >= d >= z$ (back in from $z/\sin\Psi$ to z):

$F(d)= \theta'/\pi + (\pi-2\Psi)/2\pi$

For $0 <= d <= z$ (from z to 0):

$F(d)= \theta'/\pi + (\pi-2\Psi)/\pi$

For $-z <= d <= 0$: (from 0 to -z):

$F(d)= \theta'/\pi + (\pi-\Psi)/\pi$

Clearly, since Ψ is a function of the system state, the distance distribution will also depend the initial system state.

The probability density function defined by the distance

distribution derived in equation 8 has a mode near one tail. This is the result of the system dynamics and the fact that the extreme in the distance moved does not occur at the extreme of theta. This results in two perturbed states, arrived at by different values of theta,leading to the same final state. The probability density associated with the distances of these final states is therefore greater than final states arrived at by a unique value of theta.

One consequence of this model of perturbation, with a uniformly distributed theta, is that the perturbed system is unstable. This is the result of the mode in the tail of the distance probability density which makes the expected value of distance moved nonzero. This is easily seen from equations 8 by noting that the region of enhanced probability density depends on psi. Psi in turn is a function of the ratio of population sizes which maximizes the region of enhanced probability density near the axis. Thus the closer system is to one axis (one species domination) after perturbation, the more likely it is to move even closer on the next perturbation.

A second consequence of the model is a partition of the phase space. For any point on the isocline, but one, there is a single perturbed state that return directly to the point, and the expectation of return is either toward one axis or the other. But for one point on the isocline, the expectation of return is to the original state, and this point partitions the isocline into two regions. Populations beginning in one part will tend under perturbation to drift toward one species and populations in the other part will drift towards the other species.

One outcome of the partition is to make higher intrinsic growth rates (r) selectively advantageous. Since a uniform perturbation of constant size can be considered as selecting a point at random on the circumference of the semi-circle centered on the original, pre-perturbation state, the perturbation is selectively neutral. The return trajectory which would return the system to the original state intersects this semi-circle, creating two unequal sections. The point of intersection is a function of the relative growth rates (a product of population and

intrinsic growth rate) at that point on the isocline.

The probability of a perturbation moving the system state into either section is proportional to the size (along the circumference) of that section. At each population ratio (point on isocline) the population with the larger intrinsic growth rate will have a larger chance of having the advantage than expected from population ratios. That is, random drift is toward the axis nearest the larger section, tending to favor the species with the higher than proportional growth rate.

This type of perturbation induces constraints in the lower level dynamics. For example to produce a decline in one population balanced by a growth in the other, on a time scale such that this can be regarded as a perturbation, would seem to require rather special dynamics. A more general perturbation structure is needed.

A new perturbation structure can be postulated which will generate more plausible perturbations. First the range of theta can be constrained so as to only decrease species numbers; second the generating function for the

perturbation can be modified to account for density dependence. These modifications will be made in the next section. A simulation model will also be developed to explore the behavior of the system.

Section 5

Perturbing the Competition System, a Predation Model

Under some conditions, the perturbation treatment developed in the previous section effectively tied a population decrease of one species to a population increase of the other species. This section will develop a model, based on predation, which leads to a more plausible type of perturbation. Note, that although predators may be placed at a higher hierarchical level than the prey, based on predator population dynamics, the effects of predation in a competition model can be at a lower hierarchical level.

For example a grazer species may have population dynamics which are long compared to the grazed species yet the predation (grazing) sufficiently sudden and infrequent so as to be modeled as a perturbation.

The model developed here will span a range of interactions from full density dependence to full density independence. Although the model is developed in terms of predation it may be applied to population losses due to any cause at this resolution.

The predation model is developed by considering two extremes. In density independent predation, the expected loss to a given species is independent of the population of that species. One mechanism is to consider species populations as clustered, such as might occur when fish school.

If both populations are clustered so that both have equal numbers of clusters, then population size differences would show up only in the size of the clusters. Then if the predators behave such that they forage until a cluster is discovered, and become satiated before a cluster is depleted, the expected loss for each species is related to the ratio of the number of clusters and not the population sizes.

A model that can be used for situations in which the populations are not clustered, or the predator is not satiated after consuming one cluster, can be developed as follows. Assume a predator has a requirement for a specific number of prey, the predator is indifferent as to which species the prey consist, and the predator will search for and consume clusters until it is satiated.

Then the prey are assumed to be clustered, with the number of clusters the same for both species. The cluster size for each species is proportional to the species populations. If the cluster size of either of the species is smaller than the number of prey needed to satiate the predator, then the predator will seek an additional cluster.

When the cluster size is larger than the satiation number, then the expected numbers lost to predation will be the same for both species regardless of the population sizes. This is density independent predation. If the cluster size is less than the satiation number, then the numbers lost to predation will be a function to population sizes. In the limit, where the cluster size is one individual, the losses to predation are fully density dependent.

Figure 4 illustrates the results of a model simulating this effect. The proportions of losses for different degrees of clustering can be closely approximated with the following relation:

(9) The proportion of losses sustained by species i is

$$= (0.5)^{(1.0-C_i)} \quad \{ N_i/(N_X+N_Y)\}^{Ci}$$

Where C_i is a "clustering" coefficient in the range [0,1].

Figure 4

Predation Model of Perturbation

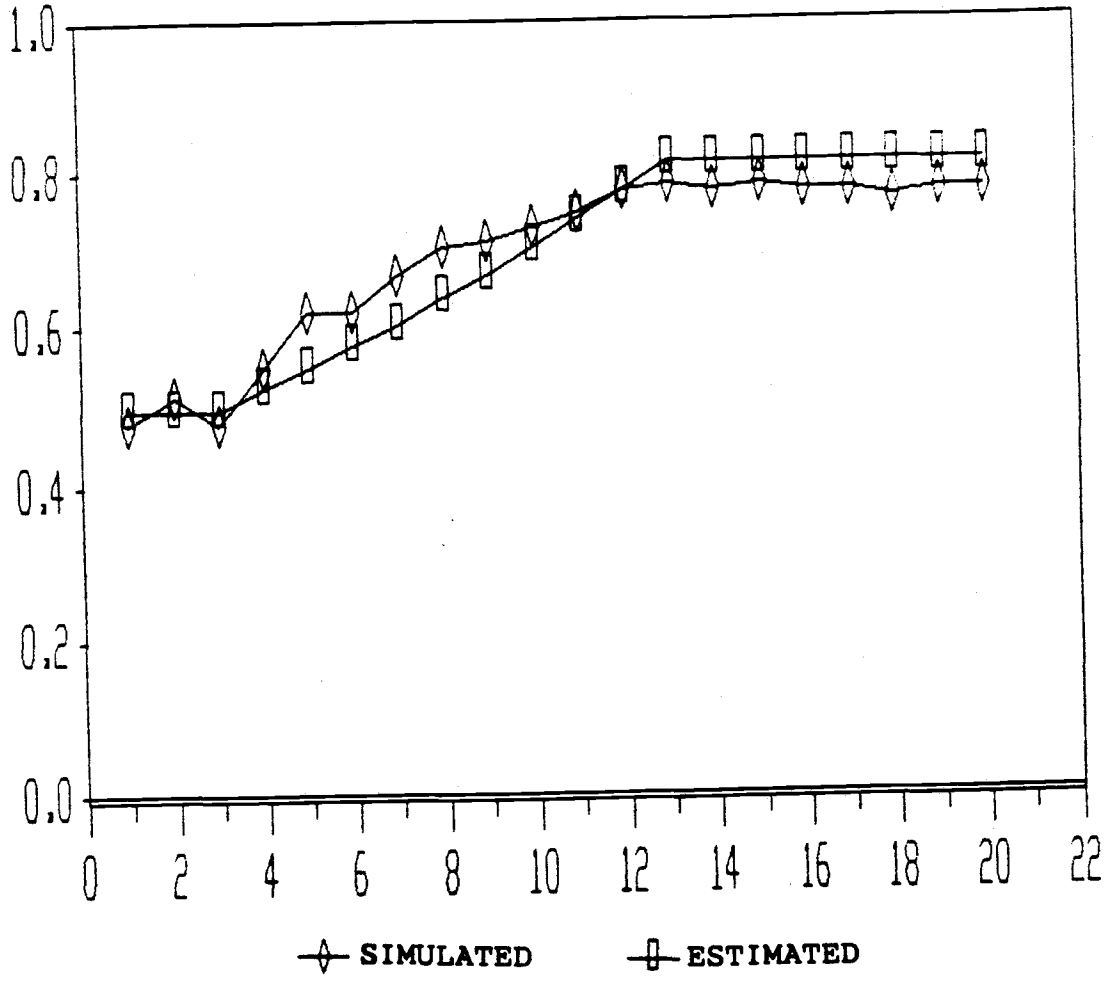


Figure 4: Predation model simulation results compared with approximation.

Simulation model with species X cluster size of 3 and species Y with a cluster size of 13. The ordinate is the proportion of the total predation loss taken from species Y. The abcissa is the total predation loss. When the total loass is less than minimum cluster size, both species contribute 50% of loss. When predation loss is larger than the largest cluster, both species lose in proportion to population.

Section 6

Behavior of Competition Model with Predation Type Perturbations

This section explores the behavior of the two species/one resource competition model when perturbed by the predation model created in the last section. Since the model is analytically intractable, a simulation is used to generate the behavior. The system dynamics are those of equation 5.

The model examines behavior near equilibrium and the equilibrium dynamics of the system are used. Perturbations are assumed rare compared to the characteristic time of the system. When this is the case the relations developed in section 2 for the distance moved on the isocline after a perturbation can be used rather than the system dynamics directly. This distance was given by equation 8.

The effects of density dependence are included in the simulation model using the relations developed in section 5 to modify the perturbation angle. The perturbation to the population is applied in three steps.

First an angle, theta, is chosen from a uniform distribution with an appropriate range. In this model the range spanned is $\pi/2$ radians and covers the third quadrant. The angle is used to project the unit perturbation vector onto each species axis. This projection is the density independent change in each population due to the perturbation.

Second, the density dependent factor of equation 9 is used to modify this change in each population as a function of its fraction of the total population. These changes are the final perturbation changes to the populations.

Finally an effective perturbation angle is calculated from the arctangent of the ratio of the perturbations. This effective angle is used when the distribution of the perturbation angles is accumulated. The entire process has the effect of filtering the original uniform distribution of angles into an empirical distribution which reflects the influence of density dependence.

The full details of the simulation model are presented in

appendix A. In brief, however, the model operates as follows. A file of parameters is first read to initialize the model. This file contains values for the type of perturbation, the growth rates for the two species, the initial equilibrium populations, the total resources available, and the density dependence of each species. Multiple sets of these parameters can be loaded simultaneously, as well as several other options discussed later.

The algorithm followed is:

- perturb each population set,
- calculate the state upon recovery,
- accumulate the effective perturbation angle and distance moved on the isocline for the summary frequency distribution,
- redistribute the population sets to the initial state.

This is repeated for as many iterations as are specified. The frequency statistics output from this model, are used by a second graphics program which formats and displays the distributions.

Figures 5 through 8 display the results of several simulation runs. Each figure is similar except for changes

in the density dependence or clustering parameters and the ratios of the growth rates.

Each figure contains a representation of the isocline of equilibrium values of the two populations. The population numbers of species X are represented on the abscissa while those of species Y are represented on the ordinate. Since both populations have been renormalized, the isocline is the hypotenuse of an isosceles triangle.

Figure 5

Distributions with C= 0.0 R ratio 1.2:0.8

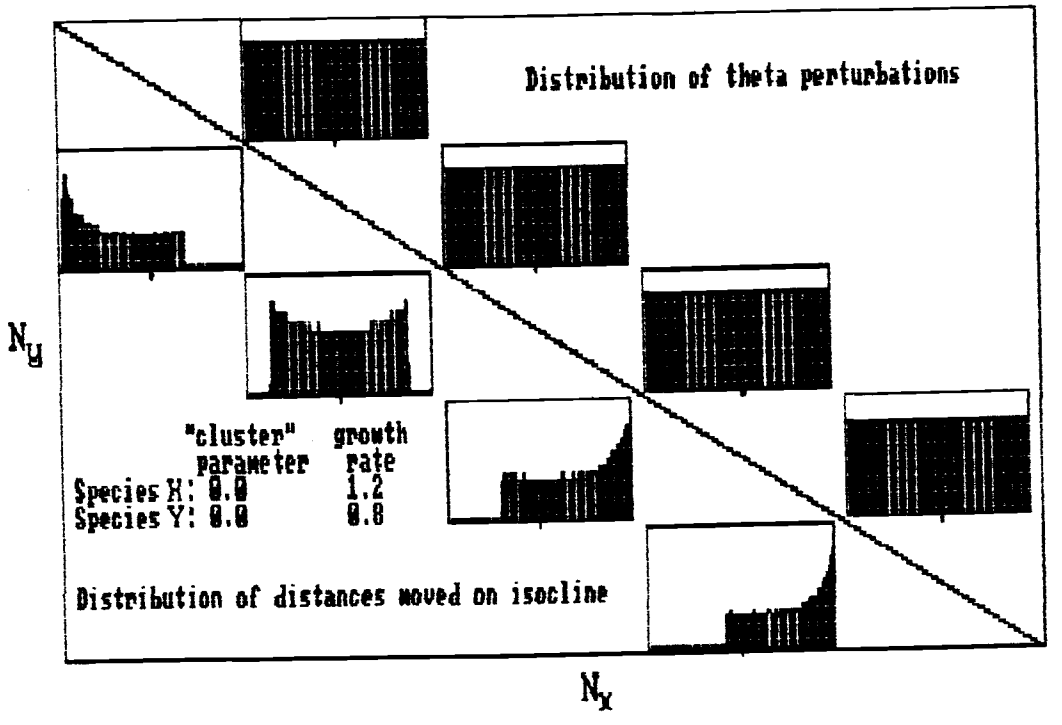


Figure 6

Distributions with C= 1.0 R ratio 1.2:0.8

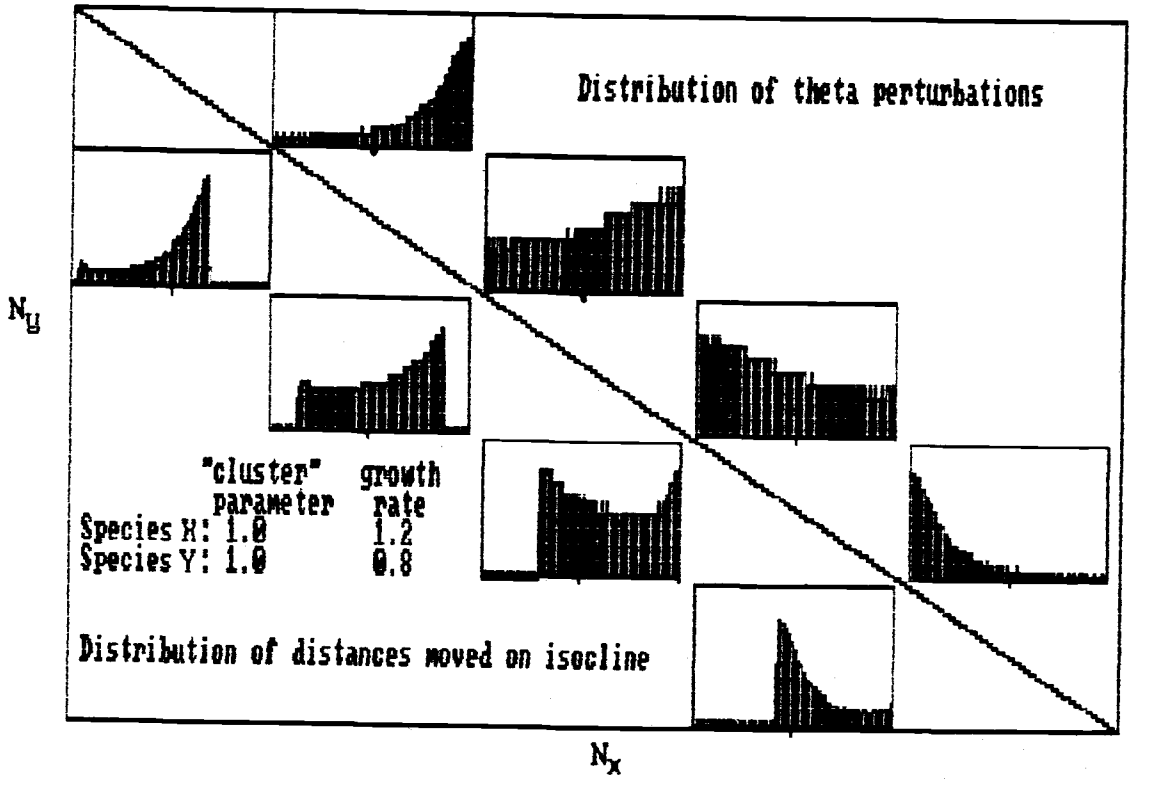


Figure 7

Distributions with C= 0.0 R ratio 1.1:0.9

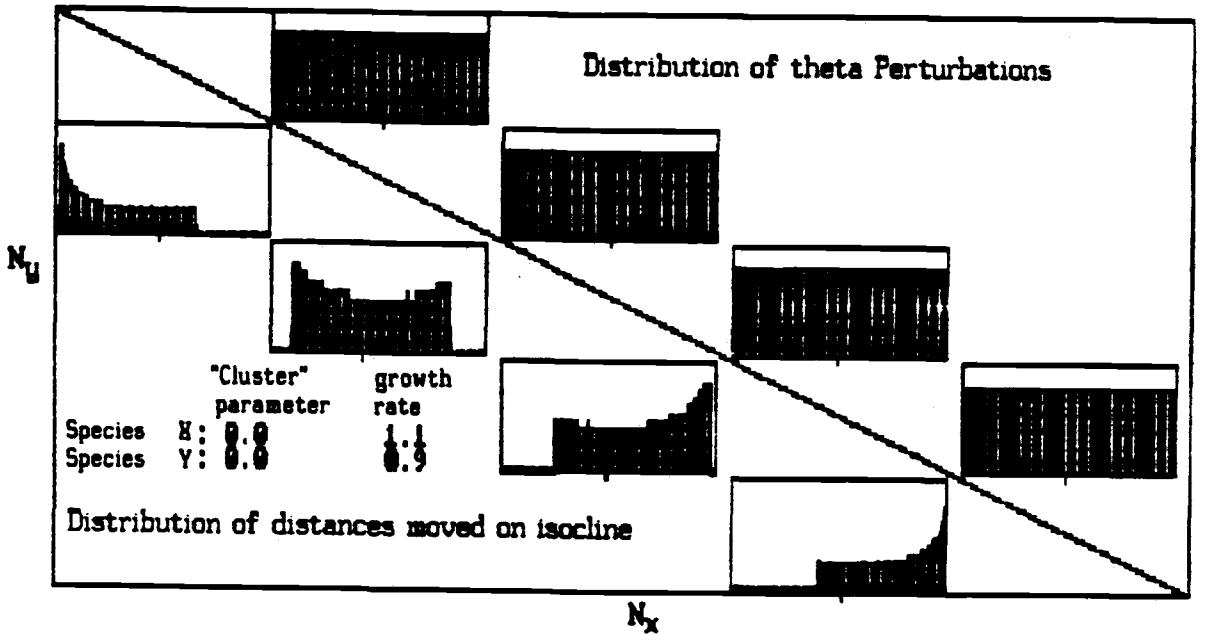
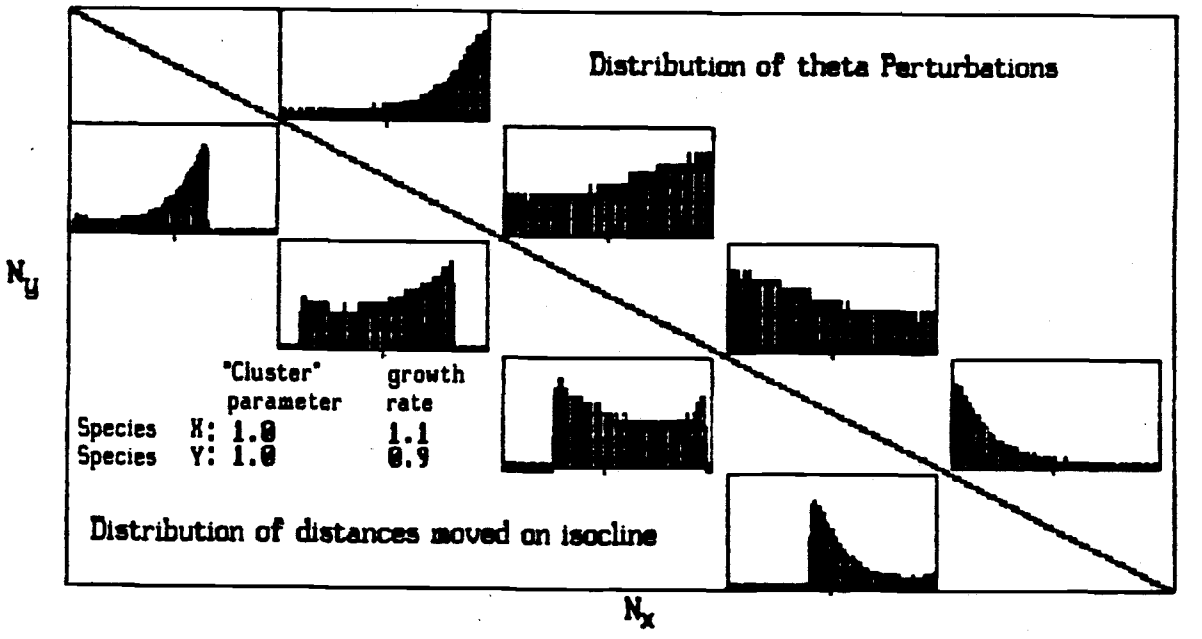


Figure 8

Distributions with C= 1.0 R ratio 1.1:0.



Along the isocline are eight histograms. The four above the isocline represent the distributions of the perturbation angles at four population ratios: (20,80),(40,60),(60,40),and (80,20). This arrangement allows the perception of the influence of population state on the distribution of perturbations.

The four histograms below the isocline represent the distributions of distances moved on the isocline in recovering from the perturbation. The pair of histograms at each population ratio may be thought of as input and output distributions of the neutrally stable competition system.

Note that each histogram spans the range of values experienced by the variables over all population ratios. Thus the perturbation angle always covers its span, while the distance moved does not. The distance moved span reflects the result of system dynamics which, for some population ratios, can result in distances moved on the isocline being greater than the perturbation magnitude.

The center of the perturbation angle span represents an orthogonal perturbation. The center of the distance moved

span represents a zero movement. From this it can be seen that for some population ratios, the expected movement upon perturbation recovery is toward the closest axis, in spite of extreme skewing of the distribution toward the equal ratio point on the isocline.

Figure 5 illustrates the behavior of the system when the ratio of the growth rates is 1.2:0.8 and there is no density dependence. In this case the distribution of perturbation angles is uniform. The distribution of distances moved is skewed toward the nearest axis for the (20,80), (60,40), and (80,20) population ratios. At (40,60), which is the ratio for which the return trajectory is orthogonal to the isocline, the distribution of recovery distances is bimodal and the expected distance moved is 0.

Figure 6 illustrates the behavior of the system with the same growth rate ratios but density dependence parameters of 1.0, corresponding to full density dependence. Note that the distributions of effective perturbation angles have become skewed. The skew of the distribution of distance moved on recovery, as seen in figure 5, has been reversed by the density dependence.

Figures 7 and 8 show the results for similar runs with the growth rate ratio of 11:9 rather than 3:2. The difference in distribution skewness between the no density dependence case and the density dependence case is the same. In fact the distribution shapes are much the same. However the range and location of the distribution of d is shifted.

In summary the shape of the distribution of d is determined by the density dependency parameters, while the location is determined by the ratio of the growth rates.

Section 7

Resource Perturbation Behavior

The previous sections explored the reaction of the system to a perturbation in the state variables of the system. This section will examine the behavior under perturbation in the resource parameter.

A perturbation to resources can occur at either of two levels. A higher hierarchical level perturbation occurs slowly compared to the population dynamics. A perturbation at a lower hierarchical level occurs quickly compared to population dynamics. In either case the notion of perturbation is maintained to connote an infrequent event such that the population is at equilibrium before each event and will reach a new equilibrium following that event.

Higher level resource perturbations, occurring with dynamics slower than the population, are tracked by the system. In the renormalized version of model 7 such changes of the resource value do not show in the phase space presentation. The renormalization compensates for changes in the scale of resources.

However a second type of perturbation can occur at the higher level time scale. This is a change in the limiting resource for which competition is occurring. There are several variants of this type of change.

The first case is when a new resource becomes limiting for the same two competing species. When this occurs the new limiting resource has a virtual isocline which sweeps through the phase space as the availability of the new critical resource declines relative to the old. When the virtual isocline intersects the original isocline it will pick up the system state and become the new isocline of the limiting resource. The appearance of the phase space of the renormalized model will not change when this is completed. There is a transition stage that will not be treated here.

A second case occurs when a new resource becomes limiting for one species but not the other. This can change the basic nature of the interaction. In one variant of this case, the phase space could be defined as in model 5 where one species competes for all of both resources while

the other competes for some of one and all of the other resource. As was discussed in section 3 this leads to exclusion of the species which does not have a resource "refuge".

A second variant of this case occurs when the new resource creates a "refuge" for both species. In this case the model is that of model 5 and has an asymptotically stable equilibrium point in the phase space. The behavior of this system is as discussed in classical competition theory and is not the subject of this inquiry.

A third case occurs when a new resource become limiting for a new competitor. In this case, one of the two species is competing with a different species for a new resource. This would create a new system at a new state in a new phase space, and a new realization of the model dynamics. There is not necessarily any relationship between the new and old realizations, and the competition relation between the two species analysed would no longer exist.

The other major type of resource perturbation is at a lower hierarchical level. A perturbation which occurs at a lower

hierarchical level is not tracked by the population dynamics. Rather the change in resource availability leaves the system state at a point in the phase space removed from the new renormalized isocline. The system will then re-equilibrate to the isocline in its own characteristic time.

Because none of the species parameters are changed, but rather only R, the relative effect of this perturbation is tho change the state variable orthogonally, either positively or negatively. Then the equilibrium will follow the specific species dynamics.

Although the effects of this type of perturbation can be handled analytically, the simulation model can be used to visualize the systems' response to this type of perturbation.

The simulation of a resource perturbation is the same as fixing the angle of the perturbation vector to two fixed values, orthogonal to the original isocline. Then the perturbation vector length is taken as a random variable from a uniform distribution.

The effect of this type of resource perturbation is to move the populations to a new state which is either above the equilibrium isocline or below it. In the first case, under reduction in the resource, both populations exceed the carrying capacity and recover by reducing the population numbers. The reduction takes place as the state follows a trajectory back to the new equilibrium.

If the perturbation increases the resource availability, this is precisely analogous to a population reduction. The populations both increase as the system state follows a trajectory back to the isocline.

For identical perturbations which raise or lower the resource availability, the respective return trajectories are near mirror images of each other.

Figure 9 illustrates the behavior of the system with this type of perturbation. The layout of the figure is the same as described in section 6. The perturbations took place at population ratios of (20,80), (40,60), (60,40) and

(80,20). Density dependency as this has no meaning for a resource perturbation model and so is not modeled. The frequency histograms for theta are not plotted, as these consisted of but two values: ($\pi/4$, $5\pi/4$). The distributions of distances moved after recovery are displayed beneath the isocline.

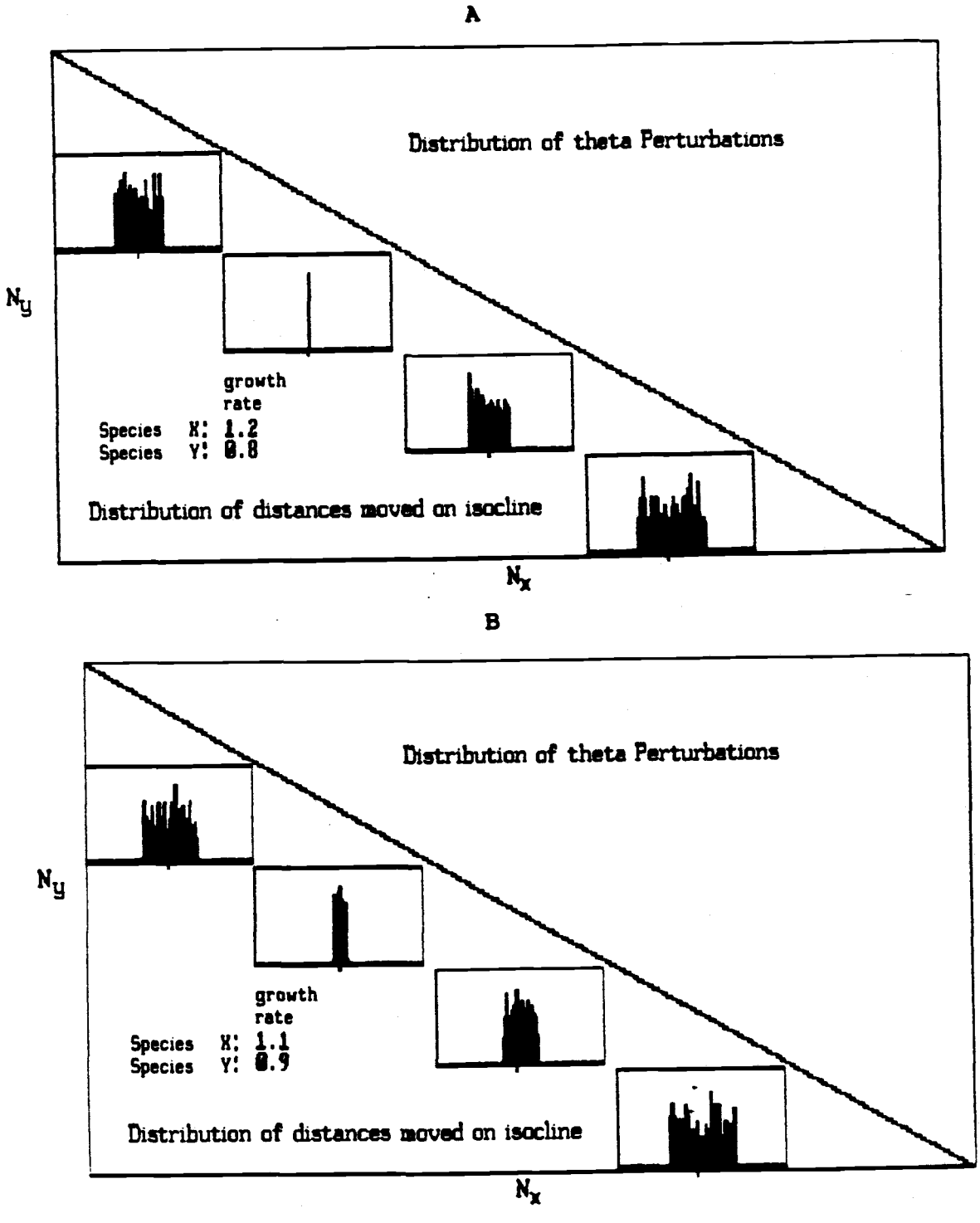
The range of the distribution is a function of the distance of the perturbation origin from the point on the isocline where the trajectories intersect orthogonally. This can be seen most clearly in the case where the growth ratios are (1.2:0.8). The orthogonality point in this case is at (40,60) which is one of the states examined. Note that at this state the range of the recovery distances is minimal (actually singular). As the states further from this point are sampled, the range of the distribution increases. In contrast when the growth rates are in the ratio of (1.1:0.9), the orthogonal point is at (45,55) not (40,60). The range of the distributions of distances moved at the sampled state (40,60) is now larger.

The median and expected values of the distribution of

distances moved in recovering from a resource perturbation is zero using the approximation of equation 8 used in the model. However, by examining figure 3 which shows the error of this approximation at one point on the isocline, one can see that perturbations above the isocline (resource decreases) always overestimate the distance moved upon recovery while perturbations below the isocline always underestimate the distance moved on the isocline upon recovery. The net effect of these biases is to cause the system to appear neutrally stable when it is actually unstable.

Figure 9

Effect of Resource Perturbation



Section 8

Population Transformation Behavior

The resource flow competition model thus far has been examined statically. The model has been perturbed at selected points in the phase space and the resultant distribution of changes in state mapped. In order to investigate more dynamic aspects of the model, an hierarchical transition is needed.

This transition involves aggregating the behavior of populations of similar system realizations and modeling the aggregate behavior. The structural level of the new model then becomes the population of system realizations rather than populations of competing species.

The simulation model will accommodate the change in focus from one level of organization to the next. Additional parameters are used from the file of simulation parameters when this change is made. These are a specification of the hierarchical level, the number of replications to use of each parameter set, the standard deviations for the population growth rates, and the standard deviations for

the density dependence parameters.

The algorithm for setting up the simulation is modified slightly when this type of run is specified. For each set of parameters loaded, the specified number of replicates of the set is generated. When the standard deviations of parameter values are nonzero, each parameter (growth and density dependence) is drawn from a Normal distribution with the specified mean and standard deviation. This generates a population of replicate systems which are similar, but not identical.

Multiple sets of populations of replicate systems can be simulated in each run. The number of systems which can be replicated per run is limited only by available computer memory.

The algorithm followed for this model level is:

- perturb each population set over all replicates
- calculate the state upon recovery
- accumulate the system states for each parameter set group for the summary frequency distributions.
- leave each population set at the recovery state.

This is repeated for as many iterations as are specified. The graphics module uses the output statistics to generate and display the distributions.

Figure 10 illustrates the output from this process. Five frequency distributions are shown here. The uppermost diagram shows the initial distribution of the population of the first species. Since at equilibrium the resources are normalized to 100 and the total species number is constant, no information is lost by presenting the distribution of just one of the species. Initially all the replicates of the competition system have the same population of 20.

Figure 10

Population Model Initial Distribution

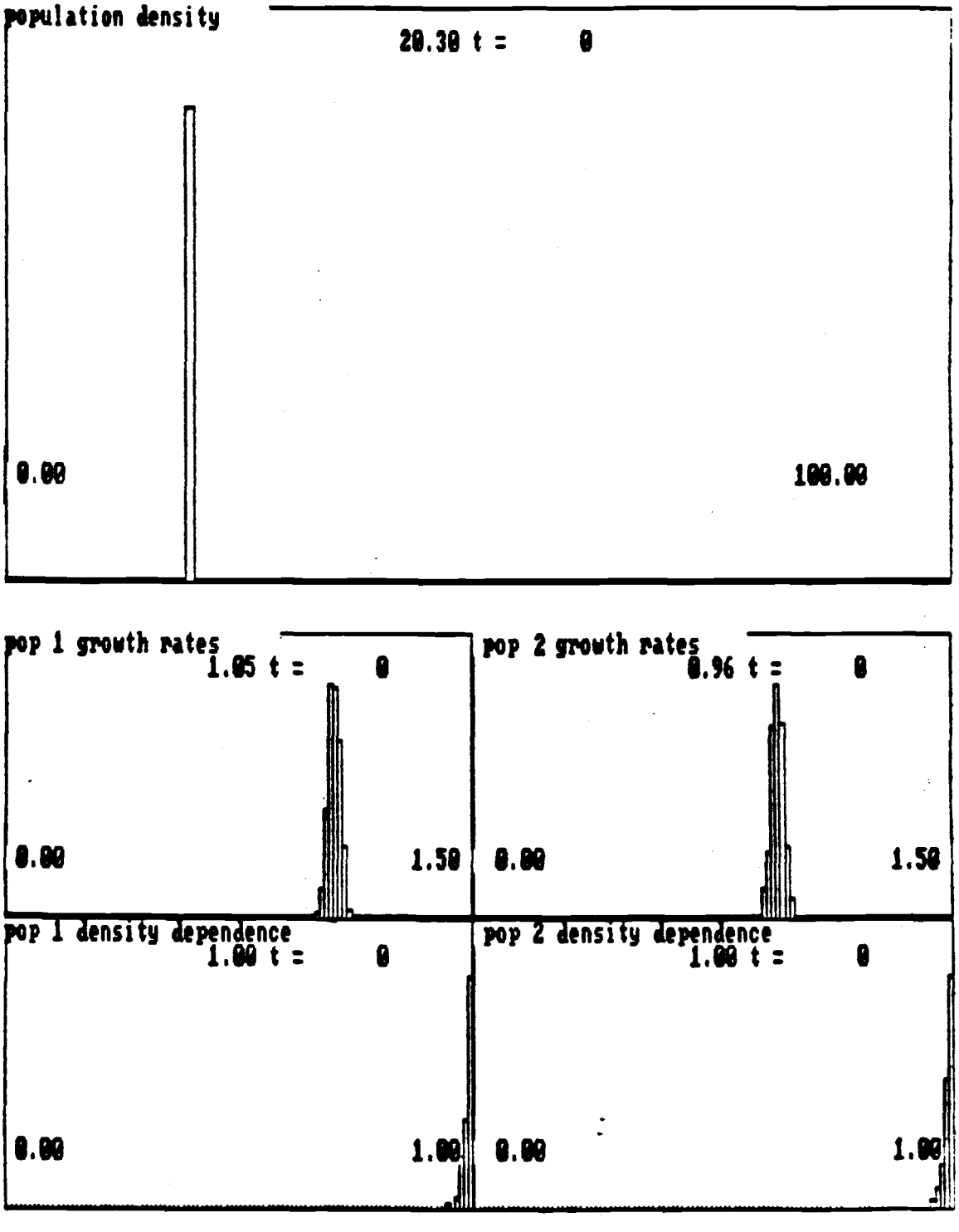
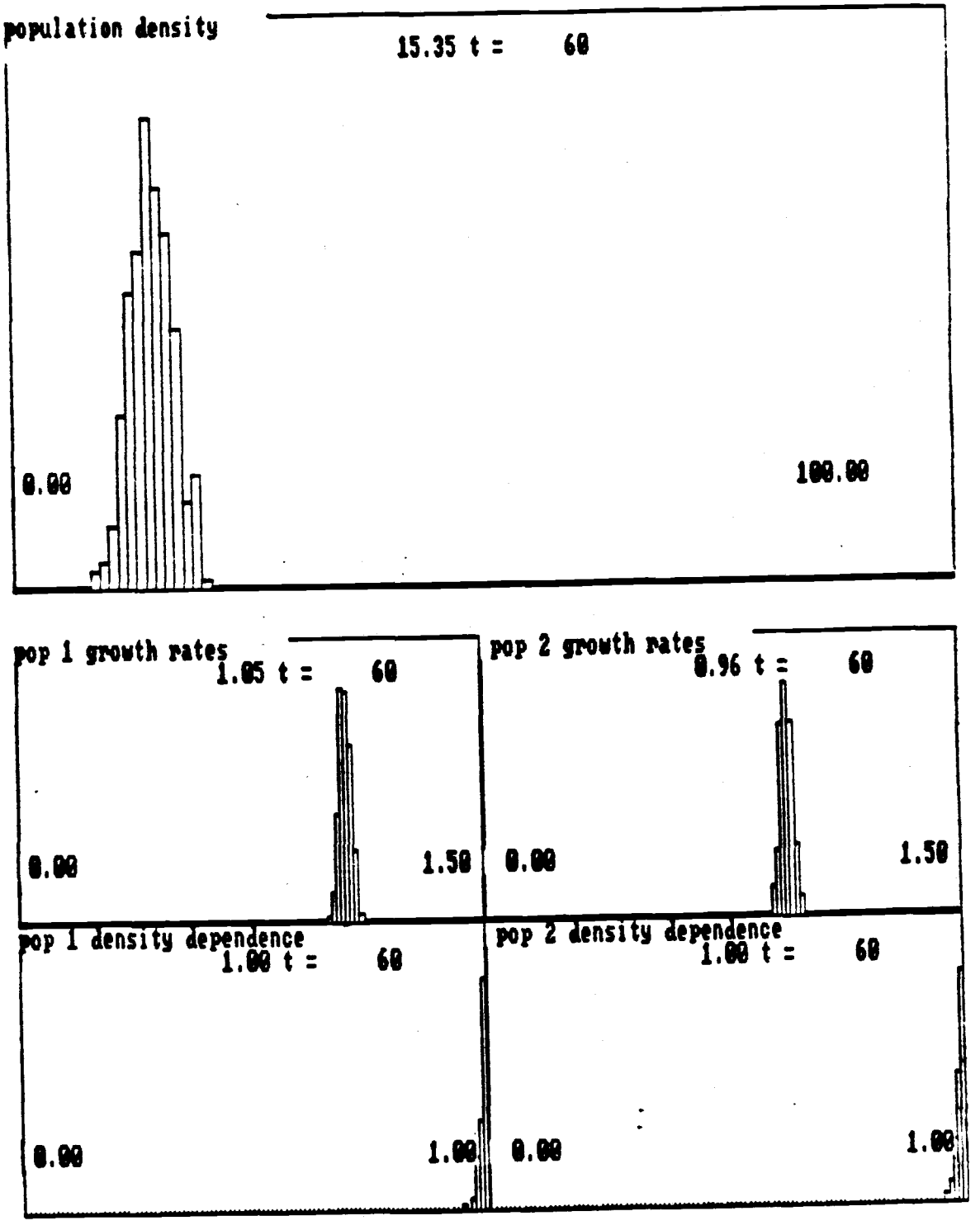


Figure 11

Population Model Distribution at T=60



Below the population distributions in the preceding figures are the distributions for the growth rates for each population and the distributions for the density dependent parameter for each population. These will not change with time. For this run of the model the growth rate ratios were 1.05:0.96. The density dependence parameter was 1.0 for each population. This is reflected in the modes of the distributions. In each case the numerical value of the distribution mode is shown in the top center of the diagram and the potential span of the parameter is shown on the left and right edges.

Figure 11 displays the same system after 60 generations. Note that the population distribution has changed in location. The mode has shifted from 20 to 15. The dispersion of the population distribution has increased as the mode has shifted. In contrast, the distributions of the parameters have stayed constant.

A more compact display of the dynamics of the system distributions is shown in figure 12. The first figure traces the change in mode of four sets of replicates. Each

set is originally at the one of the four populations examined in the static case, 20,40,60 and 80 and is identified by one of four symbols. For this simulation the growth rate ratio was 1.2:0.8. and the density dependence parameter value was 0.5 for both populations.

Figure 12a

Population Mode for C=0.5 R ratio 1.2:0.8

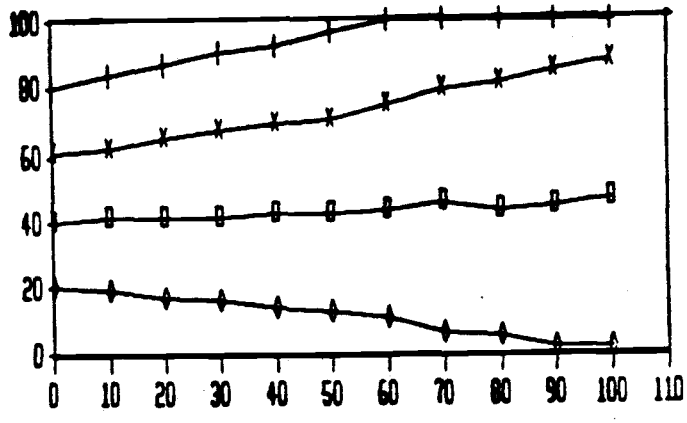
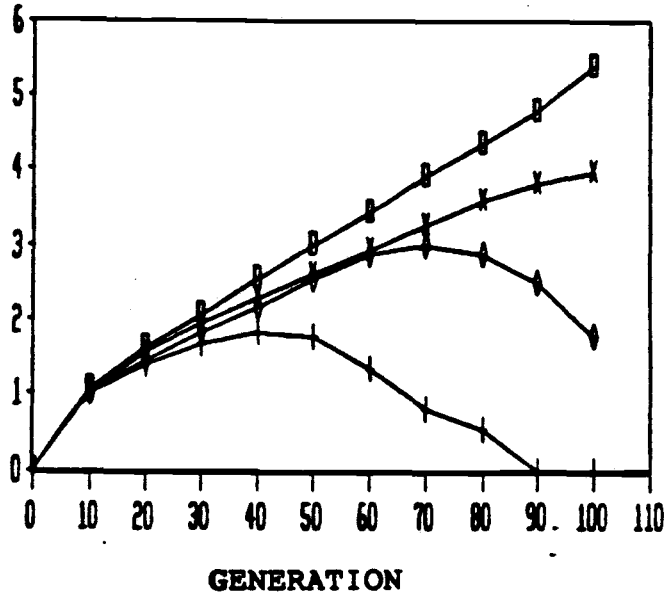


Figure 12b

Standard Deviation for C=0.5, R ratio 1.2:0.8



Three of the four modes have trajectories which lead to dominance by one species. Only the replicate originating at 40 has a relatively constant mode for a substantial number of generations. This point (40,60) on the isocline is the point where the return trajectories are orthogonal for the growth rate ratios used.

The changes in standard deviations of the distributions for each of these curves is shown in figure 12b. The symbols correspond between the two figures. As one might expect, the replicates in which one species excludes the other have a drop in the variation as the exclusion becomes complete. On the other hand, the replicate originating at a population of 40, has a variation which continues to increase. In this case, even though the mode is not changing, a larger number of population ratios will be found in the replicates as the number of generations increases.

The effect of the density dependence parameter on the model can be seen in figure 13. This figure represents model runs identical to the one represented in figure 12 except the density dependence parameter is increased from 0.5 to

1.0. In this case the movement towards dominance by one species or the other is slowed.

Figure 13a

Population Mode for C:=1.0, R ratio 1.2:0.8

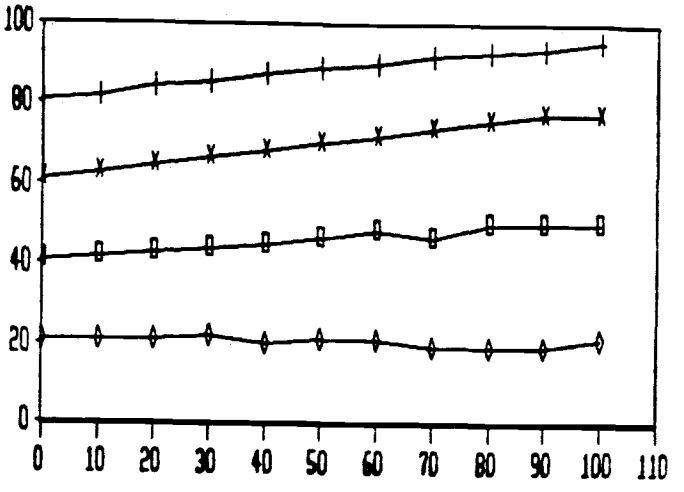
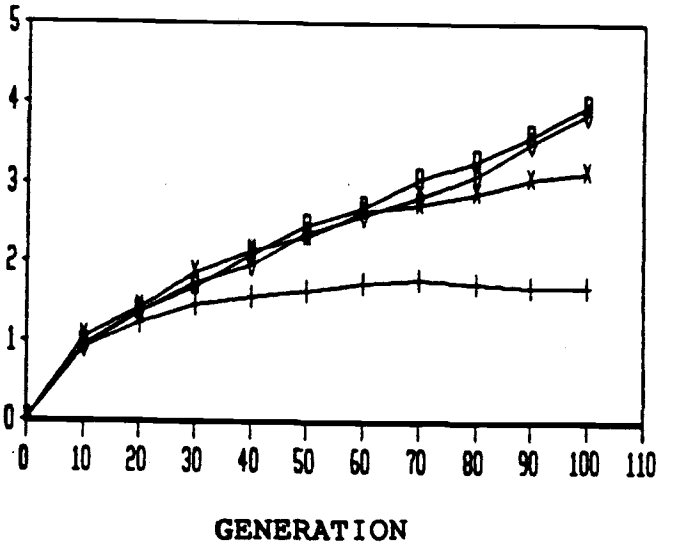


Figure 13b

Standard Deviation for C:=1.0, R ratio 1.2:0.8



In figure 13b note that the variation increases for more of the replicates than occurred with lower density dependence. This is a result of fewer of the replicates becoming dominated by one species. However the maximum standard deviation achieved by the replicate starting at population 40 is less than in figure 12. This is an effect of the lessened dispersion caused by the variation in growth rate ratios.

Figure 14 illustrates the effect of perturbation density dependence by looking at two extremes. The simulation starts from just one population in this case and uses a growth rate ratio of 1.05:0.95 to lessen the impact of the growth rate differential. Two values of density dependence, 0.0 and 1.0 are used.

Here, even with the smaller growth rate ratios, the populations with density independent perturbations rapidly are dominated by one species. The density dependent perturbations are a stabilizing factor for the mode.

Figure 14a

Effect of Density Dependence on Mode

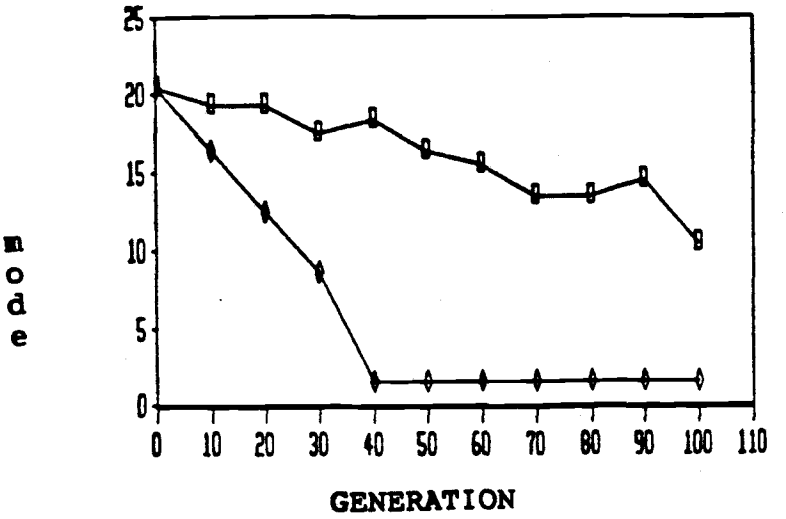
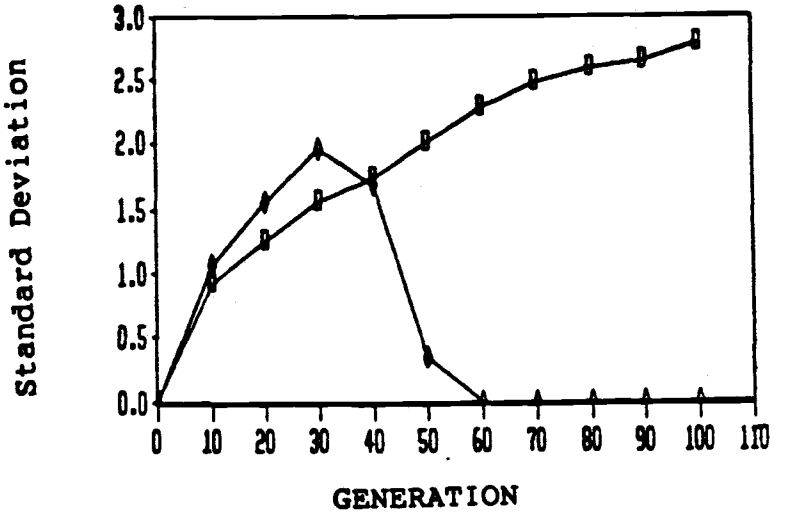


Figure 14b

Effect of Density Dependence on Dispersion



R RATIO IS 1.05:0.95

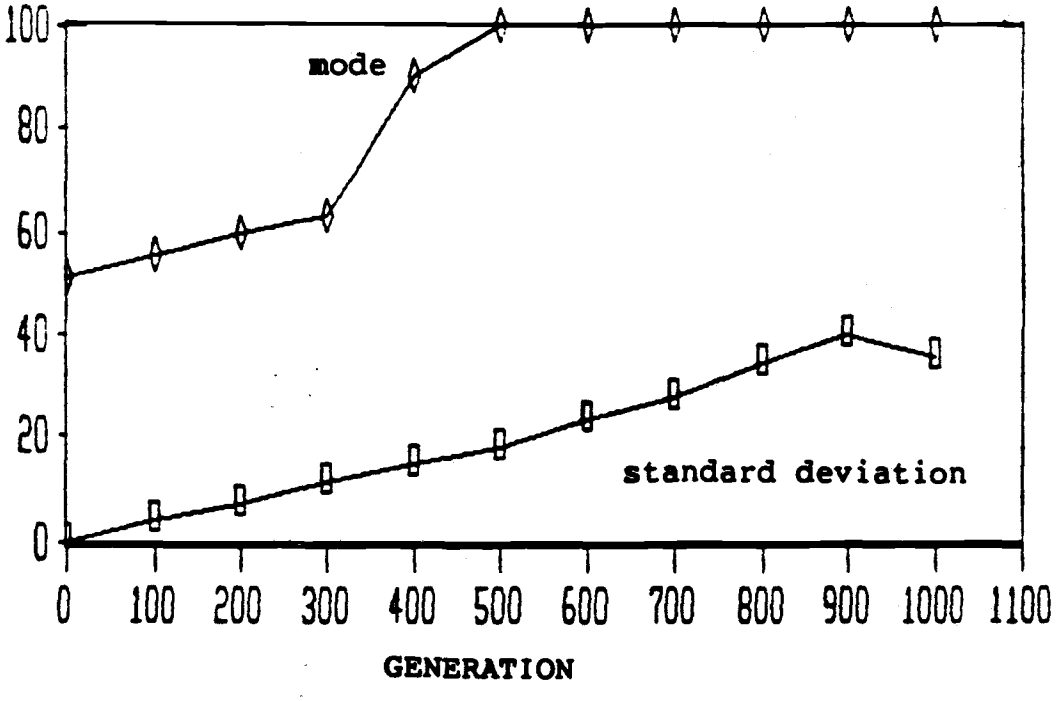
Note for the density dependent replicate, the trajectory in figure 14 can be contrasted with the trajectory with similar origin in figure 13. These have the same model parameters except for the difference in growth rate ratios. The populations in figure 13 are nearly constant while those of figure 14 are drifting toward dominance by species Y. Even though in figure 13 the initial population is close to the axis of species Y, the disparity in growth rates enables species X to maintain itself against exclusion.

Figure 14 again illustrates the increase in dispersion before exclusion. Note also the slower increase in dispersion of the replicates with the lower growth rate ratios as compared to those of figure 12b.

It appears that density dependent perturbations are a stabilizing factor in the dynamics of the model. It is not clear, however, whether a system with density dependent perturbations has a region of stability. To investigate this, a model run was set up with a 10% difference in growth rates, a ratio of 1.05:0.95, and density dependence parameter of 1.0. This was run for 1000 generations and the results shown in figure 15.

Figure 15

Long Term Behavior



r ratio 1.05:0.95, density dependence 1.0

The orthogonal trajectory state for these parameters is (47.5,52.5). The initial state for the run was chosen as both populations equal to 50.

The population of species one increased slowly for 300 generations after which it moved relatively quickly to dominance by generation 500. The standard deviation of the distribution, plotted in the lower curve, increased steadily until generation 900 after which the last of the straggler replicates began to approach dominance by species one.

It appears then that even starting a system near an expected equilibrium state, and with small disruptive forces, the eventual outcome is competitive exclusion. This is in agreement with the results of classical competition theory, but based on a different aspect of the Lotka-Volterra model.

However, since the model system is stochastic, some replicate competition systems will continue to have both species present even after most replicates have become dominated by one species or the other.

The next section will compare the simulation model to the results obtained by the classic experiments of Gauss.

Section 9

Gause's Competition Experiments

Gause (1934) performed some of the original competition experiments. Many of the experimental conditions used by Gause matched the assumptions of the model developed here. Parameter estimates from Gause's experiments were used to compare the population dynamics of the simulation model with the experimental dynamics found by Gause.

Gause performed several competition experiments which investigated different mechanisms of competition, resource as well as interference. Gause's resource competition experiments were done with two species of paramecium, Paramecium caudatum and Paramecium aurelia, feeding on one species of bacteria, Bacillus pyocyaneus. The quantity of bacteria were fixed by changing the growth media daily and adding a fixed number of bacteria.

Gause grew several replicates of each paramecium species in isolation, starting each culture with two individuals. From the measurements made on these cultures he was able to calculate the growth rate and maximum equilibrium

populations for each species. Since the size of the two species is very different, Gause translated population counts into species volumes. A P. aurelia is 0.39 of a P. caudatum in terms of volume. The values of the parameters were given in his table XI in chapter V as:

	logistic growth rates	Maximal Volume
P. aurelia	1.1244	105
P. caudatum	0.7944	64

The logistic growth rates are for volume per day in Gause's buffered 'half-loop' bacteria concentration. The 'half-loop' refers to the method of dilution of bacterial concentrations.

In analyzing a mixture of these populations, Gause realized that another conversion in terms of resource demand per unit volume was needed. The ratio can be taken directly from the table of equilibrium populations. At equilibrium each P. aurelia has the resource usage of 0.61 P. caudatum.

Gause started each culture in the competition experiment with a fixed number of bacteria as food, and two each of each species of paramecium. Both populations were allowed to grow freely, with a constant food supply until the populations were food limited (about 8 days after the culture was started). The two populations were counted every day starting on the second day. After day 8, when the species began to be resource limited, Gause removed 10% of each species population volume daily. This allowed both species to regain the equilibrium by growth from the perturbed level.

There are several differences between Gause's experiment and the simulation model. Foremost is the initial system state. Gause started his cultures with equal numbers of both species far below equilibrium. The model only deals with populations already at equilibrium. It was necessary to start the simulation at day 8 of Gause's data, when the species first approached equilibrium.

A second difference is the size and type of the perturbation. The simulation model, as discussed previously, was developed using a 1% perturbation. Gause

used a 10% density independent perturbation to his culture populations. This factor was easily changed in the model. The 10% perturbation was still within the linear range for approximating the perturbation recovery state.

The type of perturbation used is also different between Gause's experiment and the simulation model. Since he was able to count the populations, his perturbation was always exactly a 10% reduction to each species' population. In the simulation model a 10% perturbation to the total population is apportioned randomly to each species' population. For a density independent perturbation, the model does this apportioning without regard to the existing population numbers.

A third difference is a matter of scaling. Gause developed his model in terms of P. caudatum equivalents. At the end of his competition experiment the average number of P. aurelia was 350 to an average of 20 P. caudatum. In P. caudatum equivalents the total population was (350*0.39*0.61) + 20 or 103. Since the simulations were always done in terms of 100 total population members, an addition factor of (100/103) or 0.971 is needed to rescale

the populations to values comparable to those used in previous simulations.

Figure 17 shows the results of a simulation run using the parameters as estimated by Gause for his experiments. The only parameters used directly in the model are the growth rates of 1.12 for P. aurelia and 0.794 for P. caudatum. In the figure, the populations shown from the Gause data are adjusted as discussed above (P. aurelia population count * 0.39 * 0.61 * 0.971).

The simulation was started on day 8 of the Gause experiment with populations of 51 for P. aurelia and 49 for P. caudatum. The Gause data are the mean of 3 cultures. The simulated data are the mean of 200 replicates. The mean of the simulated replicates is plotted in Figure 16 along with curves of the 20th and 80th percentiles.

The actual data from the Gause experiment tends to lie above the 80th percentile for the first 7 to 8 days of the simulation. During the last few days of the experiment, the model tends to estimate values for P. aurelia close to the experimental data.

A better fit to the experimental data can be obtained if it is noted that the experimental data has a large fluctuation during the two days after the simulation is begun. If the simulation is started with the mean of the two experimental values of day 0 and day 1, 60, then the simulation matches the experimental data much more closely. This is shown in figure 17.

Figure 16
Simulation of Gauss Experiment
version 1

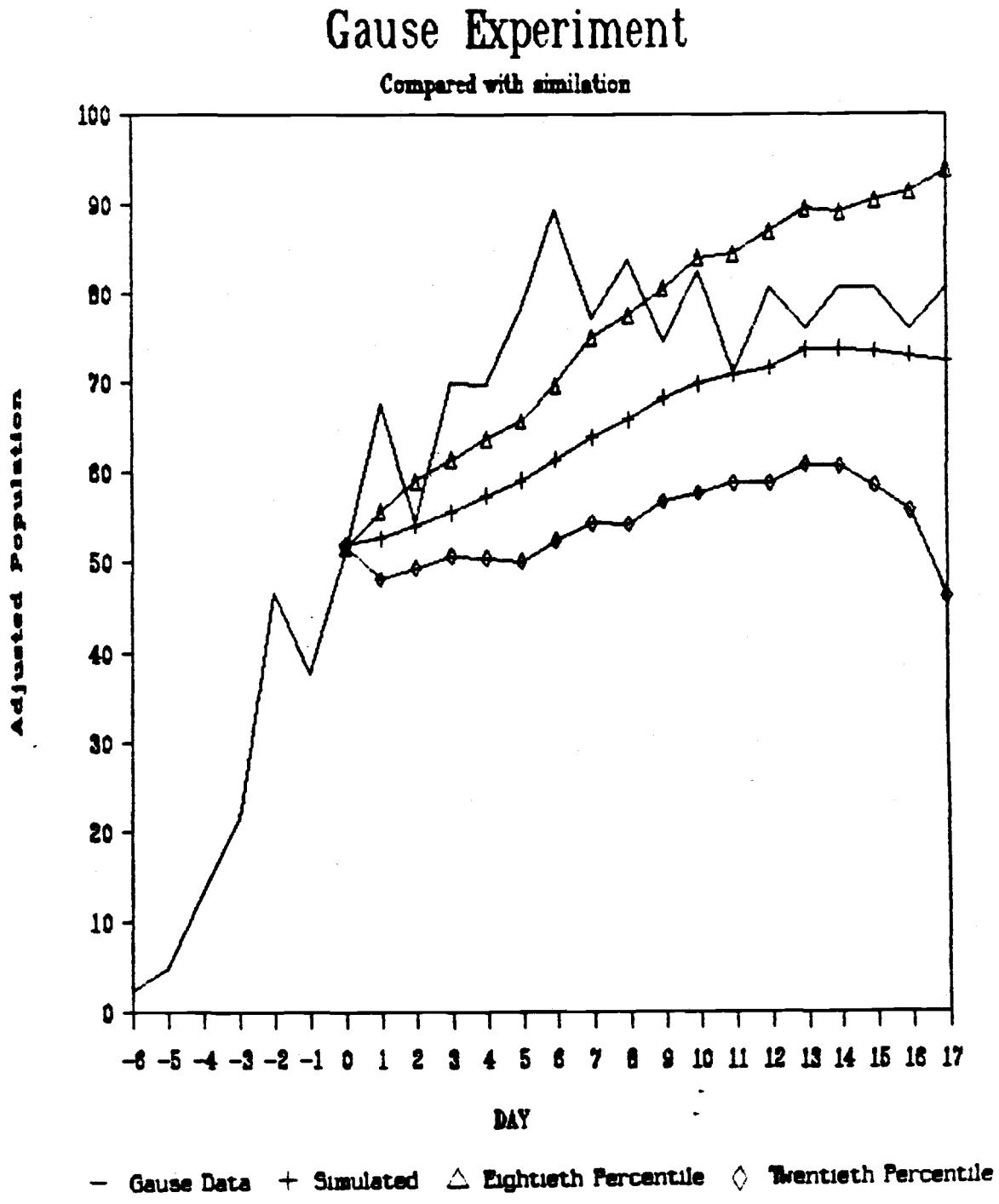
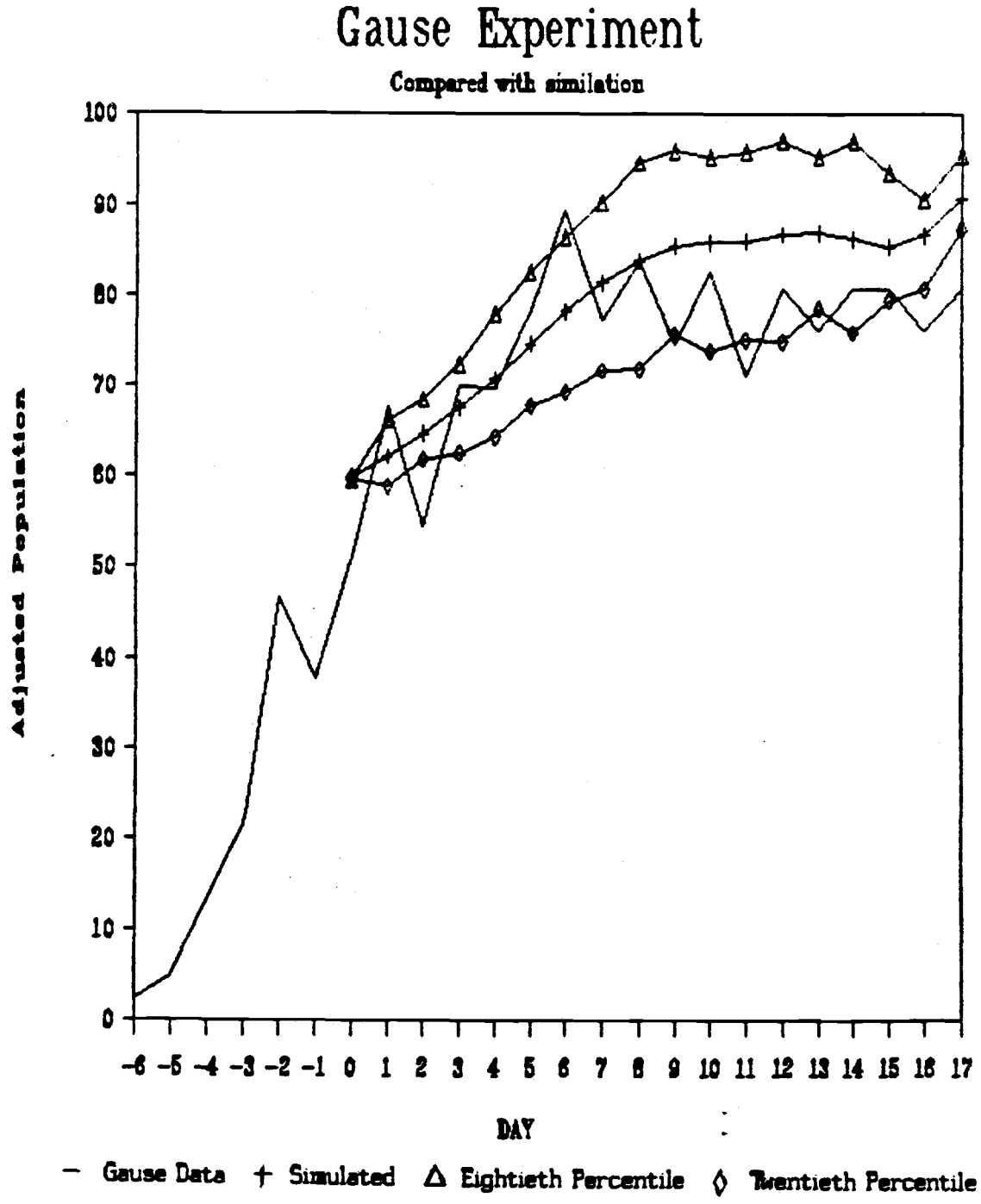


Figure 17
Simulation of Gauss Experiment
version 2



Section 10

Summary and Discussion

The goal of this research was to explore the usefulness of an resource equivalence approach to ecological models. This was approached in the context of hierarchical systems concepts which provided the framework for model development.

The Lotka-Volterra competition model was used as a starting point for the investigation. After reformulation into a resource usage form in a manner similar to that of Smith (1952), it was possible to develop a resource competition model. The resource competition model was found to have neutrally stable population dynamics for two species competing for one resource. Thus the asymptotically stable and unstable nodes seen in the classic treatment of the Lotka-Volterra model are unavailable in the two species one resource flow competition model.

Because the behavior of the neutrally stable model has been largely unexamined, several classes of perturbations were used to explore system dynamics. A static analysis produced distributions of the resultant states after

perturbations with a known distribution. A dynamical analysis tracked the changes in system statistics over time with state dependent perturbations.

Perturbations to the resource could conceptually take place at either a higher or lower hierarchical level. Lower level perturbations which generated distributions of system recovery were visualized with the simulation model. This type of perturbation to resources preserved the location of the distribution of post-perturbation system states. However the dispersion of the distribution was increased. Resource perturbation allowed the dominance of one species in the two species system, when an exact rather than an approximate recovery distribution was used.

Perturbations in the model were developed as a submodel simulating predation. This submodel created a span of perturbation types from density independent to density dependent. All population perturbations lead to one species dominating. Which species tended to dominate was the result of both the initial species ratios and the ratios of intrinsic growth rates. Density dependent perturbations retarded the rate at which dominance occurred and increased

the uncertainty in the outcome.

The simulation model results were compared with experimental data developed by Gause. Gause's experimental conditions were very similar to those assumed by the simulation model. Estimating the parameters from Gause's work, the simulation model generated population states and rates of change in those states in agreement to those reported.

As a result of the simulation, several behaviors not observed in Gause's experiment are predicted. For example, Gause always found that P. aurelia excluded P. caudatum in his cultures. However, he started all cultures with the same numbers of both species. The model predicts that if he had started the cultures with 50% more P. caudatum than P. aurelia, the P. caudatum would have won the competition. The model also predicts that if Gause had perturbed the populations either by smaller amounts, say 5%, or by a density dependent technique, the exclusion would have take place much more slowly.

Other experiments, such as Park's (1948,'52,'62,'64)

experiments with Tribolium, describe systems which do not meet the model assumptions. In Park's experiments the Tribolium engage in a high degree of interference competition.

A major result of this work is that resource flow competition systems are modeled successfully as neutrally stable dynamical systems. Rather than being uninteresting, as generally treated in the literature, these are here seen as far more interesting than the classical stable and unstable modes that are generally islands of relief from competition or clearcut competitive advantage.

The perturbation dynamics of the neutrally stable system give a theoretical understanding of the competitive system in terms of biological quantities and good predictive power. This model of resource competition has advantages over the classical treatment of the Lotka-Volterra model. In particular the model elements are clearly defined in terms of basic biological values and additional behaviors are available from the model, such as drift rates, which are not available in the classical model.

Bibliography

Connell, Joseph H. (1983) On the Prevalence and Relative Importance of Interspecific Competition. Evidence from Field Experiments. American Naturalist. Vol 122 No. 5 pp 661-696

Darwin, Charles (1859) The Origin of Species. John Murray. London

Diamond, Jared M.(1975)in Ecology and Evolution of Communities; Cody, Martin L. and Diamond, Jared M. Eds. pp 342-443 Belknap Press, Cambridge, Mass.

Fisher,Ronald A. (1930,1958) The Genetical Theory of Natural Selection. Dover, N.Y.

Frank, Peter W. (1957) CoActions in Laboratory Populations of two Species of Daphnia. Ecology Vol 38 No. 3 pp 510-519

Gause, G.F. (1934,1971) The Struggle for Existence, Dover Publications, New York, New York

Klir, George J. (1969) An Approach to General Systems Theory Van Nostrand Reinhold, New York, N.Y.

Levins, Richard (1968) Evolution in Changing Environments, Princeton University Press, Princeton, N.J.

Lotka, A.J. (1925,1956) Elements of Physical Biology. Dover Publications, New York.

Neyman, J., Thomas Park, and Elizabeth L. Scott. (1956) Struggle for Existence. The Tribolium model: Biological and Statistical Aspects. In: Proceedings of the Third Berkeley

Symposium on Mathematical Statistics and Probability. Berkely: University of California Press

MacArthur, Robert H. (1972) Geographical Ecology Harper and Row, New York, New York

May, Robert M. (1973) Stability and Complexity in Model Ecosystems. Princeton University Press, Princeton, N.J.

May, Robert M. (1976) Models for Two Interacting Populations in Theoretical Ecology, Principles and Applications. Robert M. May ed. Sinauer Associates, Sunderland, Mass. pp 78-104

Odum,Howard T. and Elisabeth C. Odum, (1976). Energy Basis for Man and Nature. McGraw Hill, New York.

Overton, W. Scott. (1977) A Strategy of Model Construction, in Ecosystem Modeling in Theory and Practice. Charles A.S. Hall and John W. Day Jr. Eds. John Wiley and Sons. New York.

Park, Thomas. (1948) Experimental Studies of Interspecies Competition I. Ecological Monographs Vol 18 no. 2 pp 266-307

Park, Thomas. (1954) Experimental Studies of Interspecies Competition II. Physiological Zoology July, Vol. 27 No. 3 pp 177-239

Park, Thomas. (1962) Beetles, Competition, and Populations. Science Vol. 138 No. 3548 pp 1369-1375

Park, Thomas; P.H. Leslie, David B. Mertz. (1964) Genetic Strains and Competition in Populations of Tribolium. Physiological Zoology, April Vol. 37 No. 2 pp 97-161

Pattee, Howard H. (1973) Hierarchy Theory: The Challenge of Complex Systems; George Braziller. New York, New York

Pianka Eric R. (1974) Evolutionary Ecology, Harper and Row. N.Y.,N.Y.

Pimm Stuart L. (1982) Food Webs. Chapman and Hall, N.Y. N.Y.

Rosen, Robert (1970) Dynamical Systems Theory in Biology, Vol I. John Wiley and Sons, New York.

Schoener, Thomas W. (1983) Field Experiments on Interspecific Competition. American Naturalist Vol 122 No. 2 pp 240-285

Simon, Herbert A. (1973) The Organization of Complex Systems in Hierarchy Theory, Howard Pattee ed. Brazziler, N.Y. pp 1-29

Slobodkin, L.B. (1961) Growth and Regulation of Animal Populations Holt, Rinehart and Winston, New York, New York

Smith, Fredrick E. (1952) Experimental Methods in Population Dynamics: a Critique. Ecology, 33:441-50.

Smith, Fredrick E. (1954) Quantitative Aspects of Population Growth, in Dynamics of Growth Processes. E. Boell ed. Princeton University Press.

Smith, J. Maynard. (1964) Models in Ecology, Cambridge University Press. Cambridge, UK

Tilman, David (1982) Resource Competition and Community Structure Princeton University Press, Princeton, N.J.

Volterra, V. (1926) Variazione e fluttuazini del numero d'individui in specie animali Conviventi. Mem. Accad. Nazionale Lincei (ser 2) 2,31-113

Whittaker, Robert H. (1975) Communities and Ecosystems. MacMillan Pub. New York.

APPENDIX

Simulation Model Source Code

```
{$g2048,p2048}
PROGRAM thesisx;
 {.he main modeling program }
 {.pw96}
 {$i c:\software\typedef.sys}

 { $ u+}                                  { allow cntrl-C to kill
                                                  program }
 { $ r+}                            { check indices at run time }
 {$V-}                              { do not check string length
                                          in procedure calls }

 {  main thesis modeling program--       }
 {   will model behavior of 2 species pure
 competition system }
 {   will also model global activity of population of
 demes   }

 {   I/O  STRUCTURE====
 }
 {      Input--  used to set parameters for a run
 }

 {   name -  ?????.PRM      ASCII
 }
 {   record #                        contents
 }
 {      1                            run name ( 35 char
 max )    }
 {      2                            file name
 }
 {      3                            # iterations,
 #demes, #dupes, selection freq }
 {      4                            simulation type,
 perturbation type, select type }
 { following set repeated for each deme type }
 {      5                            ratel,rate2,sd
 ratel,sd rate2 }
 {      6                            popl,pop2,Resource,
 sd resource }
 {      7                            C1,C2, sd c1, sd c2
 }

 {    Interchange --  used to pass run results to
 plotting and }
```

```
{                    output formating routines
}
{      two type of interchange format-- distribution,
and state }
{
}
{    distribution format
}
{     name -   ????????.DST   binary
}
{      format  record    title:string[80]
}
{                        values:array[0..maxvec] of
real      }
{
}
{    state format
}
{      name -  ????????.STT   binary
}
{       format  record           run name
}
{                                file name
}
{                                # iterations,
#demes, #dupes, }
{                                simulation type,
perturbation type, select type}
{      followed by  record(s)    #iteration,#deme,
g-rates,    }
{      (written as varient       g-sdevs,pops,
p-sdevs,cvals,  }
{       record)                   resources
}
{
}
{    output format    -- ascii
}
{        20X60 histograms and other tables
}
{        to screen , other programs create plot
files         }

{ $ i thesisx.inc}

CONST todeg = 57.29578;
```

```
 eps = 0.25;
 pturb_size = 10.0;
 smallreal = 1e-200;
 bigten = 10;
 maxvec = maxplotglb;


TYPE
 vector = ARRAY[1..maxvec] OF Real;
 title = STRING[80];


 shortstr = STRING[12];
 medstr = STRING[40];

 demepnt = ^demerec;

 demerec = RECORD
             last, next : demepnt;
             CASE Boolean OF
              True : (demeno, iterno : Integer;
                ratel, rate2, rsdevl, rsdev2,
                popl, pop2, theta, dist, psi, oldpop2,
                cval1, cval2, resource, sdev_resource,
                sdcval1, sdcval2 : Real);
              False : (run_name : medstr;
                file_name : shortstr;
                maxiter, maxdeme, dupes_per_deme :
                Integer;
                stype, ptype, seltype : Char;
                selfreq, listfreq : Integer);
            END;

 freqpnt = ^freqtype;
 freqtype = RECORD
              last, next : freqpnt;
              xmin, xstep : Real;
              maxgen : Integer;
              xtitle : title;
              sfreq : vector;
            END;


CONST maxreps = 1000;
```

```
VAR init_deme, cur_deme, tmp_deme : demepnt;
 start_freq, cur_freq, inp_freq, sys_init, sys_final :
 freqpnt;
 options : demerec;
 prmname : STRING[12];
 paramfile, outfile : Text;
 distfile : FILE OF freqtype;
 statefile : FILE OF demerec;
 iter, tmpint, badcount : Integer;
 ranvar : Real;
 nonrandom : Boolean;


 FUNCTION topower(x, pwr : Real) : Real;

 VAR i : Integer;
  tmpx : Real;
  invert : Boolean;

 BEGIN


  IF x <= 0.0 THEN
   BEGIN
    GoToXY(2, 23); Write(


     ' ------ Error: you tried to raise a negative number to a pow
    );
    Exit;
   END;


  IF pwr < 0 THEN
   BEGIN
    pwr := -1.0*pwr;
    invert := True;
   END
  ELSE invert := False;

  IF pwr < smallreal THEN
   BEGIN
    topower := 1.0;
    Exit;
```

```
   END;

  IF pwr = 1.0 THEN
BEGIN
    IF invert THEN
     topower := 1.0/x
    ELSE
     topower := x;
    Exit;
   END;

  IF (Trunc(pwr) = pwr) AND (pwr > 1.0) THEN          { if
                                                    integer
                                                  don't do
                                                     logs }
   BEGIN
    tmpx := x;
    FOR i := 1 TO (Trunc(pwr)-1) DO
     tmpx := tmpx*tmpx;

    IF invert THEN
     BEGIN
      {  if abs(tmpx)<smallreal then begin write('x is
      ',x:15:5,' pwr ',pwr:15:5,' tmpx ',tmpx);readln;
      topower:=1e307; exit; end;     }
      topower := 1.0/tmpx
     END
    ELSE
     topower := tmpx;
    Exit;
   END;



  IF pwr > 33.0 THEN
   BEGIN
    GoToXY(2, 23);
    WriteLn(
    ' --- Error: you are raising ', x:8:2, ' to the ',
    pwr:8:2,
    ' power ');
    topower := 1.0e307;
    Exit;
   END;
```

```
 tmpx := Ln(x);
 tmpx := pwr*tmpx;
 tmpx := Exp(tmpx);

 IF invert THEN
  tmpx := 1/tmpx;

 topower := tmpx;

END;


FUNCTION asin(x : Real) : Real;
BEGIN

 IF x > 1.0 THEN Exit;
 IF x < -1.0 THEN Exit;

 IF x = -1.0 THEN
  BEGIN
   asin := -1.5707963;
   Exit;
  END;
 IF x = 1.0 THEN
  BEGIN
   asin := 1.5707963;
   Exit;
  END;

 asin := ArcTan(x/Sqrt(1-Sqr(x)));

END;

FUNCTION normran(u : Real; sigma : Real) : Real;
 { function generates a random variable from a
 normal distribution with mean u and standard
 deviation sigma }

VAR u1, u2, v, tmp : Real;
BEGIN

 IF sigma < 0.0 THEN sigma := -1.0*sigma;

 IF sigma < smallreal THEN
  normran := u
```

```
 ELSE
  BEGIN
   REPEAT
    u1 := Random;
    u2 := Random;
    u1 := 2*u1-1; u2 := 2*u2-1;
    v := (u1*u1)+(u2*u2);
   UNTIL v <= 1.0;
   v := u2*Sqrt((-2.0*Ln(v))/v);

   normran := u+(sigma*v);

  END;

END;



FUNCTION distpnt(x0, y0, x1, y1 : Real) : Real;
VAR dif1, dif2 : Real;

BEGIN
 dif1 := x1-x0;
 dif2 := y1-y0;
 distpnt := Sqrt(Sqr(dif1)+Sqr(dif2));

END;

FUNCTION isocline(xpop, ypop : Real; VAR ademe :
                  demepnt) : Real;

 { Returns value of one population on isocline given
 other }

BEGIN

 IF ypop = 0.0 THEN
  BEGIN
   WITH ademe^ DO
    isocline := resource-xpop;
   Exit;
  END;

 IF xpop = 0.0 THEN
  BEGIN
```

```
   WITH ademe^ DO
   isocline := resource-ypop;
   Exit;

  END;

END;

FUNCTION ISODIST(x0, y0 : Real; VAR ademe : demepnt)
 : Real;
 { function determines distance of point from
 isocline }

CONST sqtwo = 1.414213562;

VAR m, b : Real;
BEGIN
 WITH ademe^ DO
  BEGIN
   isodist := (x0+y0-resource)/sqtwo;
  END;

END;



PROCEDURE parsefile(VAR name : shortstr; ext :
                    shortstr);

VAR i, j : Integer;

BEGIN
 i := Pos('.', name);
 IF i > 8 THEN BEGIN
  Delete(name, 9, (Length(name)-8));
  i := 0;
 END;
 IF i > 0 THEN BEGIN
  Delete(name, i, (Length(name)-i+1));
  i := 0;
 END;

 IF Length(name) < 2 THEN BEGIN
  WriteLn(' Error-- filename too short '); Halt;
 END;
```

```
name := Concat(name, '.', ext);

 END;

 PROCEDURE extend(VAR ademe : demepnt);

  { extends linked list of demes; pointer returned
  pointing to last }
  {  item on list }

 BEGIN
  IF ademe = NIL THEN
   BEGIN
    New(ademe);
    ademe^.next := NIL;
    ademe^.last := NIL;
   END
  ELSE
   BEGIN
    WHILE ademe^.next <> NIL DO
     ademe := ademe^.next;
    New(ademe^.next);
    ademe^.next^.next := NIL;
    ademe^.next^.last := ademe;
    ademe := ademe^.next;
   END;
 END;                                              { extend }

 PROCEDURE movedeme(VAR ademe, bdeme : demepnt);

  { procedure will move contents of record in ademe
  to bdeme while }
  { preserving list structure }

 VAR tmplast, tmpnext : demepnt;

 BEGIN

  tmplast := bdeme^.last;
  tmpnext := bdeme^.next;

  bdeme^ := ademe^;

  bdeme^.last := tmplast;
  bdeme^.next := tmpnext;
END;
```

```
PROCEDURE quickdisp(x, y, v, vmin, vmax : Real; ind :
                    Integer);
CONST smax = 80;
VAR i, j, k : Integer;
 scale : Real;
BEGIN
 ind := ind MOD 2;
 scale := smax/(vmax-vmin);
 GoToXY(1, (5+(ind*6)));
 WriteLn(x:15:2, y:15:2);
 WriteLn;

 k := Round((v-vmin)*scale);
 IF k > smax THEN k := smax;
 GoToXY(1, (7+(ind*6)));
 FOR i := 1 TO (k-1) DO
  Write('-');
 Write('+');
 FOR i := (k+1) TO smax DO
  Write('-');
 GoToXY(1, (8+(ind*6)));
 Write(vmin:8:0);
 GoToXY(35, (8+(ind*6)));
 Write(v:10:2);
 GoToXY((smax-10), (8+(ind*6)));
 Write(vmax:8:0);


END;



PROCEDURE initial;                    { INITIALIZES GLOBAL
                                              VARIABLES }
VAR
 tmpstr : medstr;
 i, j, thisdeme, thisiter, lastdeme, repcount :
Integer;
 done : Boolean;


 PROCEDURE zerofreq(VAR thisfreq : freqpnt);
 VAR i : Integer;
```

```
  BEGIN
   WITH thisfreq^ DO
    BEGIN
     xmin := 0.0; xstep := 0.0;
     maxgen := 0; xtitle := '';
     FOR i := 1 TO maxvec DO sfreq[i] := 0.0;
    END;

  END;


 BEGIN

  parsefile(prmname, 'prm');
  Assign(paramfile, prmname);
  Reset(paramfile);


  {assign(outfile,'debug.txt'); rewrite(outfile);}

  badcount := 0;

  WITH options DO
   BEGIN
    next := NIL; last := NIL;
    ReadLn(paramfile, run_name);
    ReadLn(paramfile, file_name);
    ReadLn(paramfile, maxiter, maxdeme, dupes_per_deme
    , selfreq,
    listfreq);
    ReadLn(paramfile, tmpstr);

    stype := ' '; ptype := ' ';

    WHILE (Length(tmpstr) > 0) AND (tmpstr[1] = ' ') DO
     Delete(tmpstr, 1, 1);
    stype := UpCase(tmpstr[1]); Delete(tmpstr, 1, 1);
    WHILE (Length(tmpstr) > 0) AND (tmpstr[1] = ' ') DO
     Delete(tmpstr, 1, 1);
    ptype := UpCase(tmpstr[1]); Delete(tmpstr, 1, 1);
WHILE (Length(tmpstr) > 0) AND (tmpstr[1] = ' ') DO
     Delete(tmpstr, 1, 1);
    seltype := UpCase(tmpstr[1]);


    WriteLn(' Options for this run ');
```

```
    WriteLn(run_name);
    WriteLn(file_name);
    WriteLn(maxiter:10, maxdeme:10, dupes_per_deme:10,
    selfreq:10, listfreq:10);
    WriteLn(stype:2, ptype:2, seltype:2);

   END;                                        { of with options }

  { now set up demes }

  init_deme := NIL;
  thisdeme := 1; thisiter := 1;


  WHILE (NOT EoF(paramfile)) AND (thisdeme <= options.
  maxdeme) DO
   BEGIN
    extend(init_deme);


    WITH init_deme^ DO
     BEGIN

      demeno := thisdeme; iterno := thisiter;
      ReadLn(paramfile, rate1, rate2, rsdev1, rsdev2);
      ReadLn(paramfile, pop1, pop2, resource,
      sdev_resource);
      theta := 0.0; psi := 0.0; dist := 0.0; oldpop2
      := pop2;
      ReadLn(paramfile, cval1, cval2, sdcval1, sdcval2)
      ;

      WriteLn(' reading deme number ', demeno:5, iterno
      :5);
      WriteLn(rate1:10:3, rate2:10:3, rsdev1:10:3,
      rsdev2:10:3);
      WriteLn(pop1:10:3, pop2:10:3, resource:10:3,
      sdev_resource:10:3
      );
WriteLn(cval1:10:3, cval2:10:3, sdcval1:10:3,
      sdcval2:10:3);

     END;                                      { with block }
```

```
IF options.dupes_per_deme > 1 THEN
 BEGIN
  cur_deme := init_deme;
  FOR i := 2 TO options.dupes_per_deme DO
   BEGIN
    extend(init_deme);          { returns pointer at new
                                                  record }

    movedeme(init_deme^.last, init_deme);          { move
                                                 contents
                                                        }

    WITH init_deme^ DO
     BEGIN
      repcount := 0;
      REPEAT
       rate1 := normran(cur_deme^.rate1, cur_deme^.
       rsdev1);
      UNTIL rate1 > 0.0;
      REPEAT
       rate2 := normran(cur_deme^.rate2, cur_deme^.
       rsdev2);
      UNTIL rate2 > 0.0;
      repcount := 0;
      REPEAT
       cval1 := normran(cur_deme^.cval1, cur_deme^.
       sdcval1);
       repcount := Succ(repcount);
       IF repcount > maxreps THEN
        BEGIN
         cval1 := cur_deme^.cval1;
         {gotoxy(1,21);write(' cval1 reps ',
         repcount:5);}
        END;
      UNTIL (cval1 >= 0.0) AND (cval1 <= 1.0);
      repcount := 0;
      REPEAT
       cval2 := normran(cur_deme^.cval2, cur_deme^.
       sdcval2);
       repcount := Succ(repcount);
       IF repcount > maxreps THEN
        BEGIN
         cval2 := cur_deme^.cval2;
         {gotoxy(1,21);write(' cval2 reps ',
         repcount:5);}
```

```
            END;

          UNTIL (cval2 >= 0.0) AND (cval2 <= 1.0);
          REPEAT
           resource := normran(cur_deme^.resource,
           cur_deme^.
           sdev_resource);
          UNTIL resource > 0.0;
         END;


       END;

     END;

    thisdeme := thisdeme+1;

   END;                                              {while loop}

  WHILE init_deme^.last <> NIL DO
   init_deme := init_deme^.last;


  cur_deme := NIL;
  done := False;

  IF options.stype = 'B' THEN
   BEGIN
    { now copy initial deme list over to active deme
    list }
    REPEAT
     extend(cur_deme);
     movedeme(init_deme, cur_deme);

     IF init_deme^.next <> NIL THEN
      init_deme := init_deme^.next
     ELSE
      done := True;

UNTIL done;

    WHILE (init_deme^.last <> NIL) AND (cur_deme^.last
    <> NIL) DO
     BEGIN
      init_deme := init_deme^.last;
```

```
      cur_deme := cur_deme^.last;
     END;
   END;

  IF options.stype IN ['S', 'P'] THEN
   cur_deme := init_deme;              { just one copy of deme
                                         list, both cur_deme }
  { and init_deme point to top of it }


  { set statistics summaries if type b(ehavior) }

  lastdeme := init_deme^.demeno;
  start_freq := NIL; cur_freq := NIL;
  inp_freq := NIL;

  IF options.stype = 'B' THEN
   BEGIN
    done := False;

    New(start_freq);
    WITH start_freq^ DO
     BEGIN
      last := NIL;
      next := NIL;
     END;
    zerofreq(start_freq);

    New(inp_freq);
    WITH inp_freq^ DO
     BEGIN
      last := NIL;
      next := NIL;
     END;
    zerofreq(inp_freq);
    REPEAT


     IF init_deme^.next <> NIL THEN
init_deme := init_deme^.next
     ELSE
      done := True;

     IF init_deme^.demeno <> lastdeme THEN
      BEGIN
```

```
    { extend list of frequencies for each deme
    type }
    New(start_freq^.next);
    start_freq^.next^.last := start_freq;
    start_freq := start_freq^.next;
    start_freq^.next := NIL;
    zerofreq(start_freq);
    New(inp_freq^.next);
    inp_freq^.next^.last := inp_freq;
    inp_freq := inp_freq^.next;
    inp_freq^.next := NIL;
    zerofreq(inp_freq);

    lastdeme := init_deme^.demeno;
   END;


 UNTIL done;


END;                                    { option is of type b}

IF options.stype IN ['S', 'P'] THEN          { setup three
                                           distributions }
BEGIN
 New(start_freq);
 cur_freq := start_freq;
 WITH start_freq^ DO
  BEGIN
   last := NIL;
   next := NIL;
  END;
 zerofreq(start_freq);

 New(inp_freq);                    { CREATED only to avoid
                                     having to test for }
 WITH inp_freq^ DO                { head pointer being nil
                                                also }
  BEGIN
   last := NIL;
   next := NIL;
  END;
 zerofreq(inp_freq);

 FOR j := 1 TO (options.maxdeme) DO
  FOR i := 1 TO 5 DO                { make 5*maxdeme total
```

```
                                                    distributions }
      BEGIN
       New(start_freq^.next);
       start_freq^.next^.last := start_freq;
       start_freq := start_freq^.next;
       start_freq^.next := NIL;
       zerofreq(start_freq);
      END;

    start_freq^.last^.next := NIL;
    Dispose(start_freq);                              { dump extra }

    start_freq := cur_freq;              { reset to top of list }
    sys_init := cur_freq;

   END;                                     { of option 's','p' }

  Close(paramfile);

  tmpstr := options.file_name;

  parsefile(tmpstr, 'STT');

  Assign(statefile, tmpstr);
  Rewrite(statefile);

  parsefile(tmpstr, 'DST');

  Assign(distfile, tmpstr);

  Rewrite(distfile);


END;                                    { initialize procedure }

PROCEDURE showstate;

VAR i, j, k : Integer;
 done : Boolean;
BEGIN
 { dump to file }

 Write(statefile, options);

 WHILE cur_deme^.last <> NIL DO cur_deme := cur_deme^.
  last;
```

```
done := False;

REPEAT

 Write(statefile, cur_deme^);
 IF cur_deme^.next <> NIL THEN
  cur_deme := cur_deme^.next
 ELSE
  done := True;

UNTIL done;

WHILE cur_deme^.last <> NIL DO cur_deme := cur_deme^.
 last;

{ write options to screen }

WITH options DO
 BEGIN

  WriteLn(' option for ', run_name);
  WriteLn(' base file name is ', file_name);
  WriteLn(' max iterations ', maxiter:5,
  ' for ', maxdeme:4, ' demes  of ', dupes_per_deme:4
  ,
  ' copies each ');
  Write(' state type is ', stype:2,
  ' perturbation type is ', ptype:2
  );
  CASE ptype OF
   'R' : WriteLn(' resource perturbation ');
   'F' : WriteLn(' full 360 theta perturbation ');
   'B' : WriteLn(' below isocline perturbation ');
   'T' : WriteLn(
         ' population decrease only perturbation ');
  END;
  CASE seltype OF
   'P' : WriteLn(' maximize population 1 ');
   'B' : WriteLn(
          ' maximize balanced population ratios ');
   'C' : WriteLn(' maximize popl density dependence '
         );
   'R' : WriteLn(' maximize popl growth rates ');
  END;
```

```
   WriteLn(

    '-------------------------------------------------',
    '----------------------');

  END;

 done := False;
 WriteLn;

 (* comment out display of state in thesisx

 repeat with cur_deme^ do begin

 writeln('  deme number ',demeno:5,' iteration ',
 iterno:5); writeln('growth rates and s.d.s ',
 rate1:10:3,rate2:10:3,rsdev1:10:3,rsdev2:10:3);
 writeln(' current populations ',pop1:10:3,
 pop2:10:3); writeln(' return angle ',
 (todeg*psi):10:3,' perturb angle ',
 (todeg*theta):10:3,' distance moved ',dist:10:3);
 writeln(' cluster paramters ',cval1:10:3,cval2:10:3,
 ' resources ',resource:10:3,sdev_resource:10:3);

 writeln; end; { with block }

 if cur_deme^.next<>nil then cur_deme:=cur_deme^.next
 else done:=true; until done;

 end comment out of state display  *)

 WHILE cur_deme^.last <> NIL DO cur_deme := cur_deme^.
  last;
 END;                                          { showstate }

PROCEDURE perturb;

 { perturb all demes }

VAR i, j, k, count : Integer;
```

```
done : Boolean;
deltap1, deltap2, sign, totpop, cfac1, cfac2 : Real;


BEGIN
 { make sure at top of list }
 WHILE cur_deme^.last <> NIL DO cur_deme := cur_deme^.
  last;
 done := False;

 count := 0;

 REPEAT

  CASE options.ptype OF

   'R' : BEGIN                              { perturbations of r }

          IF Random < 0.5 THEN sign := -1.0
          ELSE sign := 1.0;
          WITH cur_deme^ DO
           BEGIN

            resource := resource+pturb_size*Random*
            sign;
            IF sign > 0.0 THEN
             theta := Pi/4.0
            ELSE
             theta := 1.25*Pi;

            { note that in this case perturbation
            size is not }
            { pturb_size but rather isodist(pop1,
            pop2,curdeme) }

           END;

         END;

   'F' : BEGIN                                  { Full 360 theta
                                                  perturbations }

          WITH cur_deme^ DO
           BEGIN
            IF nonrandom THEN
```

```
          theta := 2*Pi*ranvar
         ELSE
          theta := 2*Pi*Random;

         totpop := pop1+pop2;
         cfac1 := topower(0.5, (1-cval1))*topower((
         pop1/totpop)
         , cval1);
         cfac2 := topower(0.5, (1-cval2))*topower((
         pop2/totpop)
         , cval2);
         deltap1 := cfac1*pturb_size*Cos(theta);
         deltap2 := cfac2*pturb_size*Sin(theta);

         IF deltap1 <> 0.0 THEN
          BEGIN
           theta := ArcTan(deltap2/deltap1);
           IF (deltap2 >= 0) AND (deltap1 < 0) THEN
            theta := theta+Pi
           ELSE IF (deltap2 < 0) AND (deltap1 > 0)
           THEN
            theta := theta+(2*Pi)
           ELSE IF (deltap2 < 0) AND (deltap1 < 0)
           THEN
            theta := theta+Pi;

          END
         ELSE
          theta := 3*Pi/2.0;

         oldpop2 := pop2;
         pop1 := pop1+deltap1;
         pop2 := pop2+deltap2;

        END;

      END;

'B' : BEGIN                          { theta below isocline }

       WITH cur_deme^ DO
        BEGIN
         IF nonrandom THEN
          theta := Pi*ranvar+(3*Pi/4)
         ELSE
          theta := Pi*Random+(3*Pi/4);
```

```
        totpop := pop1+pop2;
        cfac1 := topower(0.5, (1-cval1))*topower((
        pop1/totpop)
        , cval1);
        cfac2 := topower(0.5, (1-cval2))*topower((
        pop2/totpop)
        , cval2);
        deltap1 := cfac1*pturb_size*Cos(theta);
        deltap2 := cfac2*pturb_size*Sin(theta);


        IF deltap1 <> 0.0 THEN
         theta := ArcTan(deltap2/deltap1)
        ELSE
         theta := Pi/2.0;
        theta := theta+Pi;

        oldpop2 := pop2;
        pop1 := pop1+(deltap1);
        pop2 := pop2+(deltap2);



       END;


     END;

'T' : BEGIN                          { third quadrant theta
                                            perturbations }

      WITH cur_deme^ DO
       BEGIN
        IF nonrandom THEN
         theta := (Pi*ranvar/2.0)+Pi
        ELSE
         theta := (Pi*Random/2.0)+Pi;

        totpop := pop1+pop2;
        IF totpop < 1.0 THEN
         BEGIN
          WriteLn; WriteLn(' overflow in totpop ',
          pop1:12, ' ', pop2:12, ' ', totpop:15);
```

```
          ReadLn;
         END;
        cfac1 := topower(0.5, (1-cval1))*topower((
        pop1/totpop)
        , cval1);
        cfac2 := topower(0.5, (1-cval2))*topower((
        pop2/totpop)
        , cval2);


        deltap1 := cfac1*pturb_size*Cos(theta);
        deltap2 := cfac2*pturb_size*Sin(theta);

        { writeln(' cfac 1&2 ',cfac1:6:3,
        cfac2:6:3,' delta x  & y ',deltap1:6:3,
        deltap2:6:3);

        write(' orginal theta ',
        (theta*todeg):6:3); }

        IF deltap1 <> 0.0 THEN
         theta := ArcTan(deltap2/deltap1)
        ELSE
         theta := Pi/2.0;

        theta := theta+Pi;

        {  writeln(' new theta ',
        (theta*todeg):7:3,' new pops  ',pop1:6:2,
        pop2:6:2);}


        oldpop2 := pop2;
        pop1 := pop1+deltap1;
        pop2 := pop2+deltap2;

       END;



    END;

ELSE BEGIN
  Write(' ERROR--- invalid perturbation type ');
  Halt;
 END;
```

```
  END;                                              { case block }

  count := count+1;
  GoToXY(1, 1); Write(count:5);


  IF cur_deme^.next <> NIL THEN
   cur_deme := cur_deme^.next
  ELSE
   done := True;

 UNTIL done;

END;                                                { perturb }

PROCEDURE restore;
 { procedure will restore perturbed systems to
 isocline }
CONST ceta = 0.785398164;                { 45 degrees or pi/4}
 sqtwo = 1.414213562;

VAR done, dodist : Boolean;
 zpturb : Real;

BEGIN

 WHILE cur_deme^.last <> NIL DO cur_deme := cur_deme^.
  last;

 done := False;


 REPEAT


  IF options.ptype = 'R' THEN
   zpturb := isodist(cur_deme^.pop1, cur_deme^.pop2,
                     cur_deme)
  ELSE
   zpturb := pturb_size;

  WITH cur_deme^ DO
   BEGIN
```

```
    psi := Abs(ArcTan((rate2*pop2)/(rate1*pop1)));
    iterno := iterno+1;


    dist := zpturb*Sin(psi-theta)/Sin(ceta+psi);
    IF (theta >= psi) AND (theta < (Pi+psi)) THEN
     dist := -1.0*Abs(dist);
    IF ((theta >= (Pi+psi)) AND (theta < (2*Pi))) OR
    ((theta >= 0) AND (theta < psi)) THEN
     dist := Abs(dist);

    { WRITELN(outfile, ' distance ',dist:10:3,
    (todeg*theta):10:3,' zpturb ',zpturb:8:3); }

    pop2 := (-dist/sqtwo)+oldpop2;
    IF pop2 <= 1.0 THEN pop2 := 1.0;
    IF pop2 >= resource THEN pop2 := resource-1.0;
    pop1 := resource-pop2;



   END;                                          { with block }

  IF cur_deme^.next <> NIL THEN
   cur_deme := cur_deme^.next
  ELSE
   done := True;

 UNTIL done;

 WHILE cur_deme^.last <> NIL DO cur_deme := cur_deme^.
  last;


END;

{  procedures to accumulate statistics of input and
                                                    output }


PROCEDURE accumdist;
 { procedure will accumulate system states into
 distributions }
CONST ceta = 0.785398164;              { 45 degrees or pi/4}
```

```
VAR i, j : Integer;
 xtmp, vecrange : Real;
 done, same : Boolean;

BEGIN
 WHILE cur_deme^.last <> NIL DO cur_deme := cur_deme^.
  last;
 WHILE start_freq^.last <> NIL DO start_freq :=
  start_freq^.last;
 cur_freq := start_freq;

 done := False;
 vecrange := maxvec+1.0;

 {  ****************************************

 writeln(lst,'---------------------------- start
 accumdist  ----------------------');  writeln(lst);
 writeln(lst,' list of demes and max distances ');
 repeat with cur_deme^ do writeln(lst,' deme number ',
 demeno:3,' xmin ',cur_freq^.xmin:10:3, ' dist ',
 dist:10:3,' theta ',(todeg*theta):10:3,' psi ',
 (todeg*psi):10:3); if cur_deme^.next<>nil then
 cur_deme:=cur_deme^.next else done:=true;


 if cur_freq^.next<>nil then cur_freq:=cur_freq^.next
 else done:=true;

 until done; done:=false; while cur_deme^.last<>nil
 do cur_deme:=cur_deme^.last; while
 start_freq^.last<>nil do
 start_freq:=start_freq^.last; cur_freq:=start_freq;

 **********************    }

 REPEAT
   same := True;
  { see if first pass }

  REPEAT

   WITH cur_freq^ DO
    BEGIN
     IF xstep = 0.0 THEN
```

```
        BEGIN
         xmin := -1.0*Abs(pturb_size/Sin(ceta+cur_deme^.
         psi));
         xstep := 2.0*Abs(xmin)/(vecrange);
         maxgen := maxvec;
        END;

       I := Round(((cur_deme^.dist-cur_freq^.xmin)/
       (2.0*Abs(cur_freq^.xmin)))*maxvec)+1;
       IF (I > 0) AND (I <= maxvec) THEN
        sfreq[i] := sfreq[i]+1.0
       ELSE
        BEGIN
         WriteLn; ClrEol;

         ClrEol;
         {write(' press return to continue ');readln;}

         Halt;
         badcount := badcount+1;
        END;

       IF cur_deme^.next <> NIL THEN
        BEGIN
         cur_deme := cur_deme^.next;
        END
       ELSE
        same := False;
       IF cur_deme^.demeno <> cur_deme^.last^.demeno
       THEN
        same := False;

      END;                                          { with block }

   UNTIL (NOT same);


IF cur_freq^.next <> NIL THEN
    BEGIN
     cur_freq := cur_freq^.next;
    END
   ELSE
    done := True;
```

```
 UNTIL done;

 WHILE cur_deme^.last <> NIL DO cur_deme := cur_deme^.
  last;
 cur_freq := start_freq;


END;                                    { procedure accumstats }

PROCEDURE accuminp;
 { procedure will accumulate system states into
 distributions }
CONST ceta = 0.785398164;               { 45 degrees or pi/4}

VAR i, j : Integer;
 xtmp, vecrange : Real;
 done, same : Boolean;


BEGIN
 WHILE cur_deme^.last <> NIL DO cur_deme := cur_deme^.
  last;
 WHILE inp_freq^.last <> NIL DO inp_freq := inp_freq^.
  last;

 done := False;
 vecrange := maxvec+1.0;

 REPEAT
  same := True;
  { see if first pass }

  REPEAT
   WITH inp_freq^ DO
    BEGIN
     IF xstep = 0.0 THEN
      BEGIN
CASE options.ptype OF
        'R' : Exit;
        'F' : BEGIN
                xmin := 0.0;
                xstep := 2.0*Pi/(vecrange);
                xstep := xstep*todeg;
              END;
        'B' : BEGIN
                xmin := 2.35619449*todeg;
```

```
            xstep := Pi/(vecrange);
            xstep := xstep*todeg;
           END;
    'T' : BEGIN
            xmin := Pi*todeg;
            xstep := Pi/(2.0*(vecrange));
            xstep := xstep*todeg;
           END;
  END;                                  { of case statement }
  maxgen := maxvec;
 END;                                       { if first pass }

I := Round((((todeg*cur_deme^.theta)-xmin)/(
vecrange*xstep))*(
maxvec));

{ writeln(' xmin ',xmin:5:2,' xstep ',xstep:6:2,
' vecrange ', vecrange:6:2); writeln(' theta ',
(cur_deme^.theta*todeg):6:2,' i ',i:3, '
divisor ', ((vecrange*xstep)):8:2,' numerator ',
((todeg*cur_deme^.theta)-xmin):8:2); }


IF (i > 0) AND (i <= maxvec) THEN
 sfreq[i] := sfreq[i]+1.0
ELSE
 BEGIN
  GoToXY(1, 25);
  Write(' Theta range error ', (todeg*cur_deme^.
  theta):8:3, i
  :5,
  ' xmin ', xmin:8:2, ' xstep ', xstep:8:2);
 END;


IF cur_deme^.next <> NIL THEN
  cur_deme := cur_deme^.next
ELSE
 same := False;
IF cur_deme^.demeno <> cur_deme^.last^.demeno
THEN
 same := False;

END;                                        { with block }
```

```
  UNTIL (NOT same);

  IF inp_freq^.next <> NIL THEN
   inp_freq := inp_freq^.next
  ELSE
   done := True;

 UNTIL done;

 WHILE cur_deme^.last <> NIL DO cur_deme := cur_deme^.
  last;

 WHILE inp_freq^.last <> NIL DO inp_freq := inp_freq^.
  last;

END;                                    { procedure accuminp }


PROCEDURE showdist;
 { procedure will write out distribution files to
 intermediate }
 { format and display distributions on terminal }
VAR done : Boolean;
 number : shortstr;
 count : Integer;


BEGIN
 ClrScr;
 Rewrite(distfile);

 WHILE start_freq^.last <> NIL DO start_freq :=
  start_freq^.last;
 cur_freq := start_freq;
 WHILE inp_freq^.last <> NIL DO inp_freq := inp_freq^.
  last;
 done := False;
  number := '';
 count := 1;

 REPEAT


  WITH inp_freq^ DO
   BEGIN
    Str(count, number);
```

```
    xtitle := Concat('theta  for ', number,
    '''th deme');
    { display(con,xmin,xstep,maxgen,sfreq,xtitle); }
   END;

  Write(distfile, inp_freq^);

  {          writeln;      needed because display
  ends with a write }


  WITH cur_freq^ DO
   BEGIN
    Str(count, number);
    xtitle := Concat('distance for ', number,
    '''th deme');
    {       display(con,xmin,xstep,maxgen,sfreq,
    xtitle); }
   END;

  Write(distfile, cur_freq^);

  { writeln;       needed because display ends with a
  write }


  IF cur_freq^.next <> NIL THEN
   BEGIN
    cur_freq := cur_freq^.next;
    count := count+1;
   END
  ELSE
   done := True;
  IF inp_freq^.next <> NIL THEN
   inp_freq := inp_freq^.next
  ELSE
   done := True;

 UNTIL done;

 cur_freq := start_freq;
 WHILE inp_freq^.last <> NIL DO inp_freq := inp_freq^.
  last;

END;
```

```
PROCEDURE accumovr(first : Boolean);
 { procedure to accumulate distributions of
 populations, rates, and density dependencies for
 'S' and 'P' type models }

VAR i, j : Integer;
 done : Boolean;
 r1freq, r2freq, p1freq, c1freq, c2freq : freqpnt;
 minr, rstep, vecrange : Real;

 PROCEDURE copyfreq(VAR afreq, bfreq : freqpnt);
 VAR tlast, tnext : freqpnt;
 BEGIN
  IF (bfreq = NIL) OR (afreq = NIL) THEN
   BEGIN
    WriteLn(' overflowed freq pointer list ');
    Halt;
   END;

  tlast := bfreq^.last;
  tnext := bfreq^.next;
  bfreq^ := afreq^;
  bfreq^.next := tnext;
  bfreq^.last := tlast;

 END;

 PROCEDURE init_accum;
 VAR i, j, k : Integer;
  tmpfreq : freqpnt;
  strval : shortstr;

 BEGIN

  { make sure deme list at top }
  WHILE cur_deme^.last <> NIL DO cur_deme := cur_deme^
   .last;
  WHILE start_freq^.last <> NIL DO start_freq :=
   start_freq^.last;

  cur_freq := start_freq;

  FOR i := 1 TO 5 DO
   BEGIN
    CASE i OF
```

```
    1 : plfreq := start_freq;
    2 : rlfreq := start_freq;
    3 : r2freq := start_freq;
    4 : Clfreq := start_freq;
    5 : c2freq := start_freq;
   END;
   start_freq := start_freq^.next;
  END;

start_freq := cur_freq;

{ use first deme as base --- others were
duplicated from it }
vecrange := 1.0/(maxvec+1);

WITH cur_deme^ DO
 BEGIN

  Str(demeno:3, strval);

  plfreq^.xmin := 0;
  plfreq^.xstep := resource*vecrange;
  plfreq^.xtitle := Concat(' population density ',
  strval);
  plfreq^.maxgen := maxvec;

  rlfreq^.xmin := 0.0;
  rlfreq^.xstep := 1.5*vecrange;
  IF rlfreq^.xmin < 0.0 THEN rlfreq^.xmin := 0.0;
  rlfreq^.xtitle := Concat(' pop 1 growth rates  ',
  strval);
  rlfreq^.maxgen := maxvec;

  r2freq^.xmin := 0.0;
  r2freq^.xstep := 1.5*vecrange;
  IF r2freq^.xmin < 0.0 THEN r2freq^.xmin := 0.0;
    r2freq^.xtitle := Concat(' pop 2 growth rates ',
    strval);
  r2freq^.maxgen := maxvec;


  clfreq^.xmin := 0.0;
  clfreq^.xstep := vecrange;
  clfreq^.xtitle := Concat(
  ' pop 1 density dependence ', strval);
  clfreq^.maxgen := maxvec;
```

```
  c2freq^.xmin := 0.0;
  c2freq^.xstep := vecrange;
  c2freq^.xtitle := Concat(
  ' pop 2 density dependence ', strval);
  c2freq^.maxgen := maxvec;


 END;                                          { with cur_deme }

{ now make maxdeme copies of this }

FOR i := 1 TO ((options.maxdeme)-1) DO
 BEGIN
  k := (i+1) MOD options.maxdeme;
  IF k = 0 THEN k := options.maxdeme;
  Str(k:3, strval);

  tmpfreq := plfreq;
  plfreq := plfreq^.next^.next^.next^.next^.next;
  copyfreq(tmpfreq, plfreq);
  plfreq^.xtitle := Concat(' population density ',
  strval);

  tmpfreq := rlfreq;
  rlfreq := rlfreq^.next^.next^.next^.next^.next;
  copyfreq(tmpfreq, rlfreq);
  rlfreq^.xtitle := Concat(' pop 1 growth rates  ',
  strval);

  tmpfreq := r2freq;
  r2freq := r2freq^.next^.next^.next^.next^.next;
  copyfreq(tmpfreq, r2freq);
  r2freq^.xtitle := Concat(' pop 2 growth rates ',
  strval);

  tmpfreq := clfreq;
  clfreq := clfreq^.next^.next^.next^.next^.next;
  copyfreq(tmpfreq, clfreq);
  clfreq^.xtitle := Concat(
  ' pop 1 density dependence ', strval);

  tmpfreq := c2freq;
  c2freq := c2freq^.next^.next^.next^.next^.next;
  copyfreq(tmpfreq, c2freq);
  c2freq^.xtitle := Concat(
```

```
    ' pop 2 density dependence ', strval);

   END;
  { having initialized list, set back at begining }

  FOR i := 1 TO 5 DO
   BEGIN
    CASE i OF
     1 : plfreq := start_freq;
     2 : rlfreq := start_freq;
     3 : r2freq := start_freq;
     4 : C1freq := start_freq;
     5 : c2freq := start_freq;
    END;
    start_freq := start_freq^.next;
   END;

  start_freq := cur_freq;


 END;                                              { init_accum}

BEGIN

 tmp_deme := cur_deme;
 done := False;

 IF first THEN init_accum
 ELSE
  BEGIN

   FOR i := 1 TO 5 DO
    BEGIN
     CASE i OF
      1 : plfreq := sys_final;
      2 : rlfreq := sys_final;
      3 : r2freq := sys_final;
      4 : C1freq := sys_final;
      5 : c2freq := sys_final;
     END;
     sys_final := sys_final^.next;
    END;

  END;
```

```
FOR j := 1 TO options.maxdeme DO
 BEGIN
  FOR i := 1 TO options.dupes_per_deme DO
   BEGIN

    WITH cur_deme^ DO
     BEGIN
      i := Trunc(((popl-plfreq^.xmin)/plfreq^.xstep)+
      0.5);

      IF (i >= 1) AND (i <= maxvec) THEN
       plfreq^.sfreq[i] := plfreq^.sfreq[i]+1.0
      ELSE
       WriteLn(' error in plfreq, i is ', i:5);

      i := Trunc(((rate1-rlfreq^.xmin)/rlfreq^.xstep)
      +0.5);
      IF (i >= 1) AND (i <= maxvec) THEN
       rlfreq^.sfreq[i] := rlfreq^.sfreq[i]+1.0
      ELSE
       WriteLn(' error in rlfreq, i is ', i:5);

      i := Trunc(((rate2-r2freq^.xmin)/r2freq^.xstep)
      +0.5);
      IF (i >= 1) AND (i <= maxvec) THEN
       r2freq^.sfreq[i] := r2freq^.sfreq[i]+1.0
      ELSE
       WriteLn(' error in r2freq, i is ', i:5);

      i := Trunc(((cval1-clfreq^.xmin)/clfreq^.xstep)
      );
      IF i < 1 THEN i := 1;
      IF i > maxvec THEN i := maxvec;
        clfreq^.sfreq[i] := clfreq^.sfreq[i]+1.0;
      i := Trunc(((cval2-c2freq^.xmin)/c2freq^.xstep));
      IF i < 1 THEN i := 1;
      IF i > maxvec THEN i := maxvec;
      c2freq^.sfreq[i] := c2freq^.sfreq[i]+1.0;


     END;                                    { with cur_deme }

    IF cur_deme^.next <> NIL THEN
     cur_deme := cur_deme^.next;

   END;                          { for loop of dupe_per_deme }
```

```
   { now do next deme and its duplicates }

   p1freq := p1freq^.next^.next^.next^.next^.next;
   r1freq := r1freq^.next^.next^.next^.next^.next;
   r2freq := r2freq^.next^.next^.next^.next^.next;
   c1freq := c1freq^.next^.next^.next^.next^.next;
   c2freq := c2freq^.next^.next^.next^.next^.next;

  END;                                  { for loop of maxdeme }

 sys_final := start_freq;


 cur_deme := tmp_deme

END;                                                { accumovr }


PROCEDURE showovr(first : Boolean);

VAR i, j : Integer;
 done : Boolean;

BEGIN

 WHILE start_freq^.last <> NIL DO start_freq :=
  start_freq^.last;
 ClrScr;
 IF first THEN Rewrite(distfile);

 done := False;

 cur_freq := start_freq;

 REPEAT

  { with start_freq^ do display(con,xmin,xstep,
 maxvec,sfreq,xtitle); writeln;       }

  Write(distfile, start_freq^);
  FOR i := 1 TO maxvec DO start_freq^.sfreq[i] := 0.0;

  IF start_freq^.next <> NIL THEN
   start_freq := start_freq^.next
  ELSE
```

```
    done := True;

  UNTIL done;

  start_freq := cur_freq;

END;

FUNCTION criterion(VAR thisdeme : demepnt) : Real;
BEGIN
 WITH thisdeme^ DO
  BEGIN
   CASE options.seltype OF
    'P' : criterion := pop1;
    'B' : BEGIN
           IF (pop1 > 0.0) AND (pop2 > 0.0) THEN
            BEGIN
             IF pop1 <> pop2 THEN
              criterion := 1/Abs(pop1-pop2)
             ELSE
              criterion := 1e200;
            END
           ELSE
            criterion := 0.0;
          END;

    'C' : criterion := cval1;
    'R' : criterion := rate1;
   END;
  END;
END;

PROCEDURE findbigest(elimcount : Integer);
{ orders linked list of demes so that 'bigten'
                                              largest items }
  { are at head of list }


VAR pass, current, large : demepnt;
 lvalue : Real;
 donea, doneb : Boolean;
 count : Integer;


 PROCEDURE swapnt(VAR a, b : demepnt);
 VAR t : demepnt;
```

```
BEGIN
 t := a;
 a := b;
 b := t;
END;

PROCEDURE swaptwo(VAR a, b : demepnt);

BEGIN


 IF a^.last <> NIL THEN swapnt(a^.last^.next, b^.last
  ^.next)
 ELSE b^.last^.next := a;
 swapnt(a^.last, b^.last);
 IF b^.next <> NIL THEN swapnt(a^.next^.last, b^.next
  ^.last)
 ELSE a^.next^.last := b;
 swapnt(a^.next, b^.next);
END;

BEGIN

WHILE cur_deme^.last <> NIL DO cur_deme := cur_deme^.
 last;


pass := cur_deme;
current := cur_deme;
large := cur_deme;
lvalue := criterion(cur_deme);
donea := False;
doneb := False;
count := 1;

REPEAT

 current := pass;
 lvalue := criterion(pass);
 large := pass;
 doneb := False;


 IF current <> NIL THEN
  BEGIN
```

```
    REPEAT

      IF criterion(current) > lvalue THEN
       BEGIN
        large := current;
        lvalue := criterion(large);
       END;

      IF current^.next <> NIL THEN
       current := current^.next
      ELSE
       doneb := True;

    UNTIL doneb;

    IF large <> pass THEN swaptwo(pass, large);

    pass := large;


   END;                                    { current not nil }

  count := count+1;

  IF pass^.next <> NIL THEN
   pass := pass^.next
  ELSE
   donea := True;

 UNTIL (donea OR (count > elimcount));

 WHILE cur_deme^.last <> NIL DO cur_deme := cur_deme^.last;

END;




PROCEDURE eliminate;
 { procedure will eliminate some population in a
 type C model }
VAR i, j, count : Integer;
 last : demepnt;
```

```
 pivot : Real;

BEGIN
 WHILE cur_deme^.last <> NIL DO cur_deme := cur_deme^.
  last;


 CASE options.stype OF
  'B', 'P' : BEGIN
              GoToXY(1, 2);
              Write('no elimination ');
             END;
  'S' : IF (iter MOD options.selfreq) = 0 THEN
        BEGIN
         count := 0;
         pivot := bigten/(options.dupes_per_deme);
         IF (pivot <= 0.0) OR (pivot > 1.0) THEN
          BEGIN
           WriteLn(
           ' too few dupes for this value of bigten '
           );
           Halt;
          END;
         last := cur_deme;
         WHILE last^.next <> NIL DO
          BEGIN
           IF Random <= pivot THEN
            BEGIN             { randomly eliminate 0% of
                                                   demes }
             count := count+1;
             last^.popl := 0.0; last^.pop2 := 0.0;

            END;
           last := last^.next;
          END;                    { last now points at last
                                                 element }


         findbigest(count);        { partially order list }
        END;
 END;

END;

PROCEDURE redistribute;
 { procedure will redistribute deme populations in a
```

```
 type C model }
VAR done : Boolean;
 repcount : Integer;
 gone, top : demepnt;

BEGIN
 WHILE cur_deme^.last <> NIL DO cur_deme := cur_deme^.
  last;
 WHILE init_deme^.last <> NIL DO init_deme :=
  init_deme^.last;
 gone := cur_deme;
 done := False;

 CASE options.stype OF
  'B' : BEGIN                              { restore each deme to
                                               initial condition }
         REPEAT
          cur_deme^.pop1 := init_deme^.pop1;
          cur_deme^.pop2 := init_deme^.pop2;
          cur_deme^.resource := init_deme^.resource;
          IF cur_deme^.next <> NIL THEN
           BEGIN
            cur_deme := cur_deme^.next;
            init_deme := init_deme^.next;
           END
          ELSE
           done := True;
         UNTIL done;
        END;
     'S' : IF (iter MOD options.selfreq) = 0 THEN
           BEGIN                          { elimate procedure put
                                       largest 'bigten' items at
                                                          head }
            { of list after zeroing 'bigten' items
            Now replace these }
            { with largest }
            top := gone;
            done := False;

            REPEAT
             {with gone^ do quickdisp(pop1,pop2, pop1,
             0.0, 100.0, 0);}


             IF (gone^.pop1 <= smallreal) AND (gone^.
             pop2 <= smallreal
```

```
) THEN
 BEGIN
  movedeme(top, gone);
  WITH gone^ DO
   BEGIN
    {quickdisp(popl, pop2,popl,0.0,100.0,
    1);}
    REPEAT
     ratel := normran(gone^.ratel, gone^.
     rsdev1);
    UNTIL ratel > 0.0;
    REPEAT
     rate2 := normran(gone^.rate2, gone^.
     rsdev2);
    UNTIL rate2 > 0.0;
    repcount := 0;
    REPEAT
     cval1 := normran(gone^.cval1, gone^.
     sdcval1);
     repcount := Succ(repcount);
     IF repcount > maxreps THEN
      cval1 := gone^.cval1;
    UNTIL (cval1 >= 0.0) AND (cval1 <= 1.0)
    ;
    repcount := 0;
    REPEAT
     cval2 := normran(gone^.cval2, gone^.
     sdcval2);
     repcount := Succ(repcount);
    IF repcount > maxreps THEN
      cval2 := gone^.cval2;
    UNTIL (cval2 >= 0.0) AND (cval2 <= 1.0);
    REPEAT
     resource := normran(gone^.resource,
     gone^.
     sdev_resource);
    UNTIL resource > 0.0;
   END;

  top := top^.next;
 END;

IF gone^.next <> NIL THEN
 gone := gone^.next
ELSE
 done := True;
```

```
            UNTIL done;
           END;                                        { case `s' }
 END;                                                { case statements }

 WHILE cur_deme^.last <> NIL DO cur_deme := cur_deme^.
  last;
 WHILE init_deme^.last <> NIL DO init_deme :=
  init_deme^.last;

END;

BEGIN                                                  { main program }


 REPEAT
  Write(' Enter base name for parameter file ');
  ReadLn(prmname);
  Write(' enter 1 if sequence 0 if random ');
  ReadLn(tmpint);
UNTIL tmpint IN [0, 1];
Randomize;
IF tmpint = 1 THEN nonrandom := True
ELSE
 BEGIN
  nonrandom := False;
  Randomize;
 END;
ClrScr;
initial;
ClrScr;


CASE options.stype OF
 'S', 'P' : BEGIN
             accumovr(True);
             showovr(True);
            END;
END;
```

```
FOR iter := 1 TO options.maxiter DO
 BEGIN

  IF nonrandom THEN
   BEGIN
    ranvar := iter/options.maxiter;                    { random
                                                     replacement
                                                       variable }
    GoToXY(10, 3); WriteLn(' ranvar is ', ranvar:10:5)
    ;
   END;



  perturb;
  restore;

  CASE options.stype OF
   'B' : BEGIN
          accuminp;
          accumdist;
         END;
   'S', 'P' : IF ((iter) MOD options.listfreq) = 0
              THEN
               BEGIN
                accumovr(False);
                showovr(False);
               END;
         END;

  eliminate;
  redistribute;
  GoToXY(10, 1);
  WriteLn(' memory left ', MemAvail:6, ' iteration ',
  iter:5);

 END;

showstate;

CASE options.stype OF
 'B' : BEGIN
        showdist;
       END;
 'S', 'P' : BEGIN
             accumovr(False);
```

```
                showovr(False);
              END;
END;


Close(distfile);
Close(statefile);
{close(outfile);}
WriteLn(' all done ', MemAvail:5);

END.
```

```
{$g2048}
PROGRAM THSXPLT;

 {.pw96}
 {.d+}

 {    These are routines from the Turbo Graphix
 Toolbox, Borland Intl }
 {$i c:\software\typedef.sys}
 {$i c:\software\graphix.sys}
 {$i c:\software\kernel.sys}
 {$i c:\software\windows.sys}
 {$i c:\software\hatch.hgh}
 {$i c:\software\histogrm.hgh}
 {$i c:\software\axis.hgh}
 {$i c:\software\hardcpy.hgh}
 {$i c:\software\polygon.hgh}
 {.d-}

 { $ u+}
 { $ r+}

 { program will read in .STT and .DST files created
 by THESISX and create }
 { the histogram and beta distribution estimates on a
 text file for printing}
 { $i thesisx.inc}

CONST todeg = 57.29578;
 eps = 0.25;
 pturb_size = 10.0;
 smallreal = 1e-200;
 bigten = 10;
 maxvec = maxplotglb;


TYPE
 vector = ARRAY[1..maxvec] OF Real;
 title = STRING[80];


 shortstr = STRING[12];
 medstr = STRING[40];

 demepnt = ^demerec;
```

```
demerec = RECORD
            last, next : demepnt;
            CASE Boolean OF
             True : (demeno, iterno : Integer;
              rate1, rate2, rsdev1, rsdev2,
              pop1, pop2, theta, dist, psi, oldpop2,
              cval1, cval2, resource, sdev_resource,
              sdcval1, sdcval2 : Real);
             False : (run_name : medstr;
              file_name : shortstr;
              maxiter, maxdeme, dupes_per_deme :
              Integer;
              stype, ptype, seltype : Char;
              selfreq, listfreq : Integer);
           END;

freqpnt = ^freqtype;
freqtype = RECORD
             last, next : freqpnt;
             xmin, xstep : Real;
             maxgen : Integer;
             xtitle : title;
             sfreq : vector;
           END;


VAR init_deme, cur_deme, tmp_deme : demepnt;
 start_freq, cur_freq, inp_freq : freqpnt;
 options : demerec;
 prmname : medstr;
 outfile : Text;
 distfile : FILE OF freqtype;
 statefile : FILE OF demerec;
 iter, i, j, emode : Integer;
 screenminx, screenminy, screenmaxx, screenmaxy,
 printcount : Integer;
 axislabel : title;
 apoly : plotarray;
 betafuncvalue : Real;
 doblowup, isfirst, dohard, isoki, dobeta : Boolean;
 anschr : Char;


 PROCEDURE betaest(VAR one, two : Real; start, stop :
                   Integer; VAR xfreq :vector); FORWARD;
```

```
FUNCTION topower(x, pwr : Real) : Real;

VAR i : Integer;
 tmpx : Real;
 invert : Boolean;

BEGIN
 IF x <= 0.0 THEN
  BEGIN
   GoToXY(2, 23); Write(


    ' ------ Error: you tried to raise a negative number
       to a power ----');
   Exit;
  END;


 IF pwr < 0 THEN
  BEGIN
   pwr := -1.0*pwr;
   invert := True;
  END
 ELSE invert := False;

 IF pwr < smallreal THEN
  BEGIN
   topower := 1.0;
   Exit;
  END;

 IF pwr = 1.0 THEN
  BEGIN
   topower := x;
   Exit;
  END;


 IF pwr > 133.0 THEN
  BEGIN
   GoToXY(2, 23);
   WriteLn(' --- Error: you are raising ', x:8:2, ' to the ',
   pwr:8:2,' power ');
```

```
   topower := 1.0e300;
   Exit;
  END;


 tmpx := Ln(x);
 tmpx := pwr*tmpx;

 tmpx := Exp(tmpx);

 IF invert THEN
  tmpx := 1/tmpx;

 topower := tmpx;

END;


FUNCTION gammafunc(x : Real) : Real;
BEGIN
 IF (x > -smallreal) AND (x < smallreal) THEN
  gammafunc := 1.6e+308;                    { maximum real }

 IF x > 4.0 THEN
  gammafunc := (x-1)*gammafunc(x-1);
 IF x < 3.0 THEN
  gammafunc := gammafunc(x+1)/(x);
 IF (x >= 3.0) AND (x <= 4.0) THEN
  gammafunc := topower(x, x)*Exp(-x)*Sqrt(2*Pi/x)*
  (1.0+(1.0/(12.0*x))+(1.0/(288.0*Sqr(x)))-
  (139.0/(51840.0*Sqr(x)*x))-
  (571.0/(2488320.0*Sqr(x)*Sqr(x))));

END;                                                {gammafunc}


FUNCTION exporan(scale : Real) : Real;

VAR ul : Real;


BEGIN
 IF scale <= 0.0 THEN
  scale := -1.0*scale;
```

```
 REPEAT
  ul := Random;
 UNTIL ul > 0.0;

 exporan := -scale*Ln(ul);

END;

FUNCTION gammapdf(x, shape, scale : Real) : Real;
BEGIN

 shape := Abs(shape);
 scale := Abs(scale);

 IF x <= 0 THEN
  gammapdf := 0.0
 ELSE
  gammapdf := (topower(scale, (-shape))*topower(x, (
  shape-1.0))*
  Exp(-x/scale)/gammafunc(shape));

END;


FUNCTION betafunc(z1, z2 : Real) : Real;

BEGIN
 {
 tmp1:=gammafunc(z1);tmp2:=gammafunc(z2);tmp3:=gammafu
 nc(z1+z2); tmp4:=tmp1*tmp2/tmp3; betafunc:=tmp4;
 writeln(lst,' in betafunc z1,z2 are ',z1:12:3,
 z2:12:3); writeln(lst,' gamma z1,z2,z1+z2 ',
 tmp1:12;3,' ',tmp2:12:3, ' ',tmp3:12:3,' result ',
 tmp4:12:3);

 writeln(lst); }

 betafunc := gammafunc(z1)*gammafunc(z2)/gammafunc(z1+
 z2);

END;

FUNCTION betapdf(x, pone, ptwo : Real) : Real;
BEGIN
 pone := Abs(pone);
 ptwo := Abs(ptwo);
```

```
 IF (x <= 0.0) OR (x >= 1.0) THEN betapdf := 0.0
 ELSE
  betapdf := topower(x, (pone-1))*topower((1-x), (ptwo
  -1))/
  betafuncvalue;

END;
```

```
FUNCTION beta(one, two : Real) : Real;

 { returns beta distributed random number between 0
 and 1 }

VAR i, j, k : Integer;
 altone, altwo, tmp1, tmp2 : Real;
```

```
 FUNCTION gamma(shape, scale : Real) : Real;

 VAR xvar, yvar, xtmp, b, e, u1, u2, branch : Real;
  vvar, zvar, wvar, a, q, theta, d : Real;

  tryagain : Boolean;


 BEGIN


  IF shape <= 0.0 THEN BEGIN
   GoToXY(2, 23); Write(
  ' ------ Error: you tried to calculate a gamma with ',
   'a negative shape ----');
   Exit;
  END;

  IF shape = 1.0 THEN
```

```
 BEGIN
  xtmp := exporan(1.0);
  gamma := scale*xtmp;
  Exit;
 END;

tryagain := True;

IF shape < 1.0 THEN
 BEGIN
  e := Exp(1);
  b := (shape+e)/e;


  WHILE tryagain DO

   BEGIN

    REPEAT
     ul := Random
    UNTIL ul > 0.0;

    branch := b*ul;
    { writeln(' Branch ',branch:10:3,' b ',b:10:3,
    ' ul ', ul:10:3);}

    IF branch > 1.0 THEN
     BEGIN


      yvar := -Ln((b-branch)/shape);

      { write(' branch ',branch:10:3,' yvar ',
      yvar:10:3);}

      REPEAT
       u2 := Random
      UNTIL u2 > 0.0;

      IF u2 <= topower(yvar, (shape-1)) THEN
     BEGIN
        tryagain := False;
        gamma := scale*yvar;
        Exit;
       END;
```

```
      END                                          { branch>1.0 }
     ELSE
      BEGIN
       yvar := topower(branch, (1/shape));

       { write(' branch ',branch:10:3,' yvar ',
       yvar:10:3);}


       REPEAT
        u2 := Random
       UNTIL u2 > 0.0;

       IF u2 <= Exp(-yvar) THEN
        BEGIN
         tryagain := False;
         gamma := scale*yvar;
         Exit;
        END;

      END;                                       { branch <1.0 }

    END;                                          { while loop }

  END                                           { shape<1 branch }

ELSE

  BEGIN                                                 { shape>1}

   tryagain := True;

   a := 1.0/Sqrt(2*shape-1.0);
   b := shape-1.3862944;                                { ln 4.0 }
   q := shape+(1.0/a);
   theta := 4.5;
   d := 2.5040774;                                { 1 + ln theta }

WHILE tryagain DO
    BEGIN


     REPEAT
      ul := Random;
     UNTIL ul > 0.0;
```

```
      REPEAT
       u2 := Random;
      UNTIL u2 > 0.0;

      vvar := a*Ln(ul/(1-ul));
      yvar := shape*Exp(vvar);
      zvar := ul*ul*u2;
      wvar := b+(q*vvar)-yvar;

      IF (wvar+d-(theta*zvar)) >= 0.0 THEN
       BEGIN
        tryagain := False;
        gamma := scale*yvar;
        Exit;
       END;

      IF wvar >= Ln(zvar) THEN
       BEGIN
        tryagain := False;
        gamma := scale*yvar;
        Exit;
       END;
     END;                                          { while loop }

   END;                                          { shape>1 branch }

 END;                                            { gamma function }


BEGIN

 IF (one <= 0.0) OR (two <= 0.0) THEN
  BEGIN
   WriteLn(
    ' Error -- you attempted to generate a beta random
     value with bad parameters'     );
  Exit;
 END;



 IF (one = 1.0) AND (two = 1.0) THEN
  BEGIN
```

```
    beta := Random;
    Exit;
   END;

  IF (one = 1.0) THEN
   BEGIN
    beta := 1.0-topower(Random, (1/two));
    Exit;
   END;

  IF (two = 1.0) THEN
   BEGIN
    beta := topower(Random, (1/one));
    Exit;
   END;


  tmp1 := gamma(one, 1);
  tmp2 := gamma(two, 1);

  beta := tmp1/(tmp1+tmp2);

END;
```

```
PROCEDURE display(thiswindow : Integer; scale, xoff,
                  yoff, xmin, xstep :
                  Real;
              maxgen : Integer; x : vector; xtitle : title);

 {     This procedure will take a vector ( defined
 globally as an a array of real of length maxvec.
 Both maxvec and vector must be globally defined.


 const maxdisp=20.0;  { uses 20 lines for histogram,
 bottom 3 lines }
 { for x axis, x axis labels, and title }
```

```
VAR maxval, xmaxwidth, ymaxheight, newxmin, newymin,
 newxmax, newymax,
 pone, ptwo : Real;
 i, j, k, tmperror, factor, hatchdens, fcount, start,
 stop : Integer;
 aplt, apoly : plotarray;
 hatch : Boolean;
 ch : Char;

 { procedure will display vectors up to maxvec
 elements long, if  vector is}
 { half or less of that length, the procedure will
 widen number of  columns}
 { of histogram to make  a more pleasing display }

 PROCEDURE blowup;

 BEGIN
  copyscreen;
  ClearScreen;



  definewindow(1, 0, 0, xmaxglb, ymaxglb);
  defineworld(2, xmin, Round(1.2*maxval), ((xstep*
  maxgen)+xmin), 0);
  { backwards for histograms?}
  selectworld(2);
  selectwindow(1);

  setheaderon;
  setheadertobottom;
defineheader(1,
   ' expanded view of empirical histogram ');

  drawborder;
  drawhistogram(aplt, maxgen, hatch, hatchdens);
  defineworld(2, 0, Round(1.2*maxval), 1.0, 0);
  selectworld(2);
  IF dobeta THEN
   BEGIN
    selectwindow(1);
    setlinestyle(1);
    drawborder;
```

```
    drawpolygon(apoly, 1, -maxgen, 0, 2, 0);
    setlinestyle(0);
   END;
  Delay(3000);


  selectscreen(2);
  copyscreen;
  selectscreen(1);

 END;                                                    {blowup}

BEGIN


 maxval := -9.9e20;

 FOR i := 1 TO maxgen DO
  IF x[i] > maxval THEN
   maxval := x[i];

 IF (maxval > -1.0e-20) AND (maxval < 1.0e-20) THEN
  Exit;

 { writeln(' maxval is ',maxval:10:3,' maxgen is ',
 maxgen:6); }

 {copyscreen; leavegraphic;}

 FOR i := 1 TO maxgen DO
  BEGIN
   aplt[i, 1] := 0.0;
   aplt[i, 2] := x[i];
   {   write(x[i]:9:2); if (i mod 8)=0 then writeln;}
  END;



 xmaxwidth := screenmaxx-screenminx+1;
 ymaxheight := screenmaxy-screenminy+1;


 definewindow(1, Trunc(xoff*xmaxwidth+1), Trunc(yoff*
 ymaxheight),
 Trunc((scale+xoff)*xmaxwidth), Trunc((scale+yoff)*
 ymaxheight));
```

```
tmperror := geterrorcode;
IF tmperror >= 0 THEN
 BEGIN
  copyscreen;
  leavegraphic;
  WriteLn(' geterrorcode returned ', tmperror:4,
  geterrorcode:4);
  WriteLn(
  ' call to definewindow used following parameters '
  );
  WriteLn(1:5, Trunc(xoff*xmaxwidth+1):5, Trunc(yoff*
  ymaxheight):5,
  Trunc((scale+xoff)*xmaxwidth):5, Trunc((scale+yoff)
  *ymaxheight):5);
  WriteLn(' xoff was ', xoff:8:2, ' y off is ', yoff:
  8:2, ' scale ',
  scale:8:2);
  WriteLn('screen vars are (xmax ymax xmin ymin ) ',
  screenmaxx:10, ' ', screenmaxy:10, ' ', screenminx:
  10,
  ' ', screenminy:10);
  Write(' press return to continue'); ReadLn;
  entergraphic;
  selectscreen(2);
  copyscreen;
  selectscreen(1);
 END;


{defineheader(1,xtitle);}

defineworld(2, xmin, Round(1.2*maxval), ((xstep*
             maxgen)+xmin), 0);
{ backwards for histograms?}
tmperror := geterrorcode;
IF tmperror >= 0 THEN
 BEGIN
  copyscreen;
  leavegraphic;
  WriteLn(' geterrorcode returned ', tmperror:4,
  geterrorcode);
  WriteLn(
  ' call to defineworld used following parameters ')
  ;
```

```
  WriteLn(thiswindow:5, xmin:10:2, Round(1.2*maxval):
  8,
  ((xstep*maxgen)+xmin):10:2, 0:5);

  Write(' press return to continue'); ReadLn;
  entergraphic;
  selectscreen(2);
  copyscreen;
  selectscreen(1);
 END;


selectworld(2);


selectwindow(1);

drawborder;

{ drawaxis(-5,5,0,0,0,0,0,0,false);}

k := 1;
IF Trunc((scale+yoff)*ymaxheight-k) >= Trunc((scale+
yoff)*ymaxheight)
THEN k := 2;


definewindow(1, Trunc(xoff*xmaxwidth+1), Trunc(yoff*
ymaxheight),
Trunc((scale+xoff)*xmaxwidth), Trunc(((scale+yoff)*
ymaxheight)-k));

tmperror := geterrorcode;
IF tmperror >= 0 THEN
 BEGIN
  copyscreen;
  leavegraphic;
  WriteLn(' geterrorcode returned ', tmperror:4,
  geterrorcode:4);
  WriteLn(
  ' call to definewindow used following parameters '
  );
  WriteLn(1:5, Trunc(xoff*xmaxwidth+1):5, Trunc(yoff*
  ymaxheight):5,
  Trunc((scale+xoff)*xmaxwidth):5, Trunc((scale+yoff)
  *ymaxheight):5);
```

```
  WriteLn(' xoff was ', xoff:8:2, ' y off is ', yoff:
  8:2, ' scale ',
  scale:8:2);
  WriteLn('screen vars are (xmax ymax xmin ymin ) ',
  screenmaxx:10, ' ', screenmaxy:10, ' ', screenminx:
  10,
  ' ', screenminy:10);
  Write(' press return to continue'); ReadLn;
  entergraphic;
  selectscreen(2);
  copyscreen;
  selectscreen(1);
 END;


defineworld(2, xmin, Round(1.2*maxval), ((xstep*
maxgen)+xmin), 0);
{ backwards for histograms?}
selectworld(2);
selectwindow(1);

hatchdens := 7;
hatch := False;


drawhistogram(aplt, maxgen, hatch, hatchdens);



start := 1; stop := 1;
FOR i := 1 TO maxvec DO
 BEGIN
  IF (x[i] < 1.0) AND (stop = 1) THEN start := i;
  IF x[i] > 0.0 THEN stop := i;
 END;


IF dobeta THEN
 BEGIN
  betaest(pone, ptwo, start, stop, x);

  betafuncvalue := betafunc(pone, ptwo);

  fcount := 0;
  FOR i := 1 TO maxgen DO
   BEGIN
```

```
      apoly[i, 1] := i/(maxgen+1);
      IF (i < start) OR (i > stop) THEN
       apoly[i, 2] := 0.0
      ELSE
       BEGIN
        fcount := fcount+Round(aplt[i, 2]);
        apoly[i, 2] := betapdf(apoly[i, 1], pone, ptwo)
        ;
        {apoly[i,1]:=apoly[i,1]-apoly[start,1];}
       END;
     END;

    FOR i := start TO stop DO
     BEGIN
      apoly[i, 2] := apoly[i, 2]*fcount/(stop-start+1);
     END;
    (* copyscreen; leavegraphic; clrscr; writeln('beta
    parameters are: ',pone:10:3,ptwo:10:3,' count ',
    fcount:5); for i:=1 to maxgen do begin if (i mod
    5)=0 then writeln; write(i:4,apoly[i,1]:5:1,
    apoly[i,2]:5:2); end; readln; entergraphic;
    selectscreen(2); copyscreen; selectscreen(1); *)

    defineworld(2, 0, Round(1.2*maxval), 1, 0);
    selectworld(2);
    setlinestyle(1);
    selectwindow(1);
    drawpolygon(apoly, 1, -maxgen, 0, 2, 0);
    setlinestyle(0);
   END;                                          { dobeta }


  IF doblowup THEN
   BEGIN
    Delay(1000);
    blowup;
   END;



END;


FUNCTION pullreal(VAR line : title) : Real;
VAR atmp : STRING[20];
```

```
 ILOC, error : Integer;
 xrslt : Real;

BEGIN

 WHILE (NOT(line[1] IN ['0', '1', '2', '3', '4', '5',
 '6', '7', '8', '9'
 , '+'
 , '-', '.']))
 AND (Length(line) > 0) DO Delete(line, 1, 1);

 IF Length(line) = 0 THEN
  BEGIN
   pullreal := 0.0;
   Exit;
  END;


 iloc := Pos(' ', line);
 IF (iloc > 0) AND (Length(line) > 0) THEN
  BEGIN
   atmp := Copy(line, 1, (iloc-1));
   Delete(line, 1, iloc);


  END
 ELSE
  BEGIN
   atmp := line;
   line := '';
  END;

 IF Length(atmp) > 0 THEN
  Val(atmp, xrslt, error);
 IF error = 0 THEN pullreal := xrslt
 ELSE
  BEGIN
   pullreal := 0.0;
   WriteLn(' Error - no number found in string: ',
   line);
  END;


END;
```

```
CONST
 maxtable = 122;

VAR
 Betable : ARRAY[1..maxtable, 1..4] OF Real;
 initbeta : Boolean;


 PROCEDURE betaest;                          { forward declared }
  {procedure betaest(var  one,two:real; start,
  stop:integer; var xfreq:vector);   forward;}

  { procedure estimates beta distribution parameters
  given a frequency distribution which is frequency
  of values between 0.0 and  1.0}
  { uses global variable betable which is table of
  estimates }

 VAR i, j, k, lowpos, highpos : Integer;

  table : Text;

  g1, g2, step, small1, small2, large1, large2 : Real;
  altone, altwo, altg1, altg2, vals1, vals2, valb1,
  valb2 : Real;
  COUNT, last, span : Real;
  backflag : Boolean;

  FUNCTION arcextrap(low1, low2, high1, high2, mid1 :
                     Real) : Real;
   { this function will extrapolate between two table
   point with }
   { an arctangent extrapolation }

  VAR ratio, tmp1, tmp2 : Real;

  BEGIN
   ratio := (mid1-low1)/(high1-low1);
   IF ratio = 0.0 THEN
    BEGIN
     arcextrap := low2;
     Exit;
    END;
```

```
  IF ratio = 1.0 THEN
   BEGIN
    arcextrap := high2;
    Exit;
   END;

  tmp2 := ArcTan(low2);
  tmp1 := tmp2+(ratio*(ArcTan(high2)-tmp2));
  arcextrap := Sin(tmp1)/Cos(tmp1);

 END;


BEGIN

 IF NOT initbeta THEN
  BEGIN
   {writeln;writeln(' reading table ');}

   INITBETA := True;

   Assign(table, 'BETAPRM.dat');
   Reset(table);
   FOR i := 1 TO maxtable DO
    BEGIN
     ReadLn(table, betable[i, 1], betable[i, 2],
     betable[i, 3],
     betable[i, 4]);
     {            writeln(lst,i:4,betable[i,1]:10:4,
     betable[i, 2]:10:4, betable[i,3]:10:4, betable[i,
     4]:10:4);}

    END;
   Close(table);

  END;

 g1 := 0.0;
 g2 := 0.0;
 COUNT := 0.0;
 altg1 := 0.0;
 altg2 := 0.0;

 span := (stop-start+1.0);

 FOR i := start TO stop DO
```

```
 BEGIN
  step := ((i-start)+0.5)/span;
  count := count+xfreq[i];
  g1 := g1+(Ln(step)*xfreq[i]);
  g2 := g2+(Ln(1.0-step)*xfreq[i]);

  { writeln(' value ',step:8:4,' g1 ',g1:8:3,' freq
  ',xfreq[i]:4:0, ' g2 ',g2:8:3);}

 END;

{ write(' sum g1,g2 ',g1:8:3,g2:8:3);}

IF count > 0.0 THEN
 BEGIN
  g1 := g1/count;
  g2 := g2/count;
 END
ELSE
 BEGIN
  ClrScr;
  WriteLn(' bad array passed betaest - terminated ');
  Halt;
 END;
g1 := Exp(g1);
g2 := Exp(g2);

{writeln;writeln(' final g1 ',g1:10:3,' final g2 ',
g2:10:3);}



{ now find these in betable }

backflag := False;

IF g1 <= g2 THEN
 BEGIN
  altg1 := g1;
  altg2 := g2;
  backflag := False;
 END
ELSE
 BEGIN
  altg1 := g2;
  altg2 := g1;
```

```
  backflag := True;
 END;


IF altg1 < betable[1, 1] THEN altg1 := betable[1, 1];
IF altg2 < betable[1, 2] THEN altg2 := betable[1, 2];
IF altg1 > betable[maxtable, 1] THEN altg1 := betable
 [maxtable, 1];

lowpos := Trunc(100.0*altg1) DIV 5;

small1 := lowpos/20;
large1 := small1+0.05;

IF small1 < 0.05 THEN
 BEGIN
  small1 := 0.01;                { first table entry, not 0}
  large1 := 0.05;
 END;

IF small1 > 0.45 THEN small1 := 0.49;          { largest gl
                                                      value }


IF large1 > 0.45 THEN large1 := 0.49;

i := 1;


{ writeln(' small1 ',small1:10:3,' large1 ',
large1:10:3);}

{ writeln(' altg1 ',altg1:10:3,' altg2 ',
altg2:10:3); }



WHILE (betable[i, 1] < small1) AND (i < maxtable) DO
 i := i+1;

last := betable[i, 1];

{ write(' i ',i:3,' table values '); for k:=1 to 4
do write(betable[i,k]:8:3); writeln(' last ',
last:8:3); }
```

```
WHILE ((betable[i, 2] < altg2) AND (betable[i, 1] =
last)
AND (i < maxtable)) DO i := i+1;

{ write(' i ',i:3,' table values '); for k:=1 to 4
do write(betable[i,k]:8:3); writeln(' last ',
last:8:3);}

IF betable[i, 1] <> last THEN
 BEGIN
  I := i-1;
  altg2 := betable[i, 2];

  { writeln(' BACKING UP '); write(' i-1 ',(i-1):3,
  ' table values '); for k:=1 to 4 do
  write(betable[(i-1),k]:8:3); writeln; write(' i ',
  i:3,' table values '); for k:=1 to 4 do
  write(betable[i,k]:8:3); writeln(' last ',
  last:8:3);}

 END;

IF i > 1 THEN
 BEGIN
  j := i-1;

  IF (betable[i, 1] = betable[j, 1]) THEN
   BEGIN
    vals1 := arcextrap(betable[j, 2], betable[j, 3],
    betable[i, 2],
    betable[i, 3], altg2);
    vals2 := arcextrap(betable[j, 2], betable[j, 4],
    betable[i, 2],
    betable[i, 4], altg2);
   END
  ELSE
   BEGIN
    vals1 := betable[i, 3];
    vals2 := betable[i, 4];
   END;
 END
ELSE
 BEGIN
  vals1 := betable[i, 3];
  vals2 := betable[i, 4];
 END;
```

```
{ writeln(' vals1 ',vals1:10:3,' vals2 ',
vals2:10:3); }


WHILE (betable[i, 1] < large1) AND (i < maxtable) DO
 i := i+1;


last := betable[i, 1];



{write(' i ',i:3,' table values '); for k:=1 to 4
do write(betable[i,k]:8:3); writeln;}


WHILE (betable[i, 2] < altg2) AND (betable[i, 1] =
last)
AND (i < maxtable) DO i := i+1;

{ write(' i ',i:3,' table values '); for k:=1 to 4
do write(betable[i,k]:8:3); writeln(' last ',
last:8:3); }


IF betable[i, 1] <> last THEN
 BEGIN
  i := i-1;
  altg2 := betable[i, 2];

  {   writeln(' BACKING UP '); write(' i-1 ',
  (i-1):3,' table values '); for k:=1 to 4 do
  write(betable[(i-1),k]:8:3); writeln; write(' i ',
  i:3,' table values '); for k:=1 to 4 do
  write(betable[i,k]:8:3); writeln(' last ',
  last:8:3); }

 END;


IF i > 1 THEN
 BEGIN
  j := i-1;

  IF (betable[i, 1] = betable[j, 1]) THEN
```

```
    BEGIN
     valb1 := arcextrap(betable[j, 2], betable[j, 3],
     betable[i, 2],
     betable[i, 3], altg2);
     valb2 := arcextrap(betable[j, 2], betable[j, 4],
     betable[i, 2],
     betable[i, 4], altg2);
    END
   ELSE
    BEGIN
     valb1 := betable[i, 3];
     valb2 := betable[i, 4];
    END;
  END
 ELSE
  BEGIN
   valb1 := betable[i, 3];
   valb2 := betable[i, 4];
  END;

 { writeln(' valb1 ',valb1:10:3,' valb2 ',
 valb2:10:3); }


 altone := arcextrap(small1, vals1, large1, valb1,
 altg1);
 altwo := arcextrap(small1, vals2, large1, valb2,
 altg1);

 IF backflag THEN
  BEGIN
   one := altwo;
   two := altone;
  END
 ELSE
  BEGIN
   one := altone;
   two := altwo;
  END;


END;
```

```
PROCEDURE parsefile(VAR name : medstr; ext : shortstr)
 ;

VAR i, j : Integer;

BEGIN
 i := Pos('.', name);
 IF i > 0 THEN BEGIN
  Delete(name, i, (Length(name)-i+1));
  i := 0;
 END;

 IF Length(name) < 2 THEN BEGIN
  WriteLn(' Error-- filename too short '); Halt;
 END;

 name := Concat(name, '.', ext);
END;

PROCEDURE extend(VAR ademe : demepnt);

 { extends linked list of demes; pointer returned
 pointing to last }
 {  item on list }

BEGIN
 IF ademe = NIL THEN
  BEGIN
   New(ademe);
   ademe^.next := NIL;
   ademe^.last := NIL;
  END
 ELSE
  BEGIN
   WHILE ademe^.next <> NIL DO
    ademe := ademe^.next;
   New(ademe^.next);
   ademe^.next^.next := NIL;
   ademe^.next^.last := ademe;
   ademe := ademe^.next;
  END;
END;                                                { extend }

PROCEDURE extenf(VAR afreq : freqpnt);
```

```
 { extends linked list of freqs; pointer returned
 pointing to last }
 {  item on list }

BEGIN
 IF afreq = NIL THEN
  BEGIN
   New(afreq);
   afreq^.next := NIL;
   afreq^.last := NIL;
  END
 ELSE
  BEGIN
   WHILE afreq^.next <> NIL DO
    afreq := afreq^.next;
   New(afreq^.next);
   afreq^.next^.next := NIL;
   afreq^.next^.last := afreq;
   afreq := afreq^.next;
  END;
END;                                                { extenf }

PROCEDURE movedeme(VAR ademe, bdeme : demepnt);

 { procedure will move contents of record in ademe
 to bdeme while }
 { preserving list structure }

VAR tmplast, tmpnext : demepnt;

BEGIN

 tmplast := bdeme^.last;
 tmpnext := bdeme^.next;

 bdeme^ := ademe^;

 bdeme^.last := tmplast;
 bdeme^.next := tmpnext;

END;

PROCEDURE loadfreq;
VAR i, j, k, count : Integer;
 fmpnext, fmplast : freqpnt;
 tmpfreq : freqtype;
```

```
BEGIN
 count := 0;
 IF cur_freq <> NIL THEN
  BEGIN
   WHILE cur_freq^.next <> NIL DO
    cur_freq := cur_freq^.next;
   WHILE cur_freq^.last <> NIL DO
    BEGIN
     cur_freq := cur_freq^.last;
     Dispose(cur_freq^.next);
     cur_freq^.next := NIL;
    END;
   Dispose(cur_freq);
   cur_freq := NIL;
  END;


 WHILE (count < (5*options.maxdeme)) AND (NOT EoF(
 distfile)) DO
  BEGIN
   extenf(cur_freq);
   Read(distfile, tmpfreq);
   count := count+1;

   { WITH tmpfreq DO BEGIN WriteLn(' for ', count, '
   distribution xmin,xstep is ', xmin:10:3,
   xstep:10:3); WriteLn(' maxgen ', maxgen, ' ',
   xtitle); FOR i := 1 TO maxvec DO BEGIN
   Write(sfreq[i]:5:0); IF ((i MOD 15) = 0) AND (i >
   0) THEN WriteLn; END; WriteLn; IF tmpfreq.maxgen =
   0 THEN tmpfreq.maxgen := maxvec;

   END;
   WITH tmpfreq}


   fmpnext := cur_freq^.next;
   fmplast := cur_freq^.last;
   cur_freq^ := tmpfreq;
   cur_freq^.next := fmpnext;
   cur_freq^.last := fmplast;

  END;
```

```
 WHILE cur_freq^.last <> NIL DO cur_freq := cur_freq^.
  last;

END;

PROCEDURE initialize;

VAR dmpnext, dmplast : demepnt;
 fmpnext, fmplast : freqpnt;
 tmpdeme : demerec;
 tmpfreq : freqtype;
 count, i : Integer;

BEGIN

 parsefile(prmname, 'stt');
 Assign(statefile, prmname);
 WriteLn(' opening state file ', prmname);
 {$i-}
 Reset(statefile);
 IF IOResult <> 0 THEN
  BEGIN
   WriteLn(' io error in ', prmname);
   Halt;
  END;

 {$i+}
cur_deme := NIL;
Read(statefile, options);
WITH options DO
   BEGIN
    WriteLn(' found options for ', run_name, ' ',
    file_name);
    WriteLn(' maxiter ', maxiter, ' maxdeme ', maxdeme
    , ' dupes ',dupes_per_deme, ' types ', stype,
         ' ', ptype);
    WriteLn(' list frequency ', listfreq);
   END;
IF options.stype IN ['B', 'b'] THEN
   dobeta := True
  ELSE
   dobeta := False;
count := 0;
WHILE NOT EoF(statefile) DO
   BEGIN
    extend(cur_deme);
```

```
    Read(statefile, tmpdeme);
dmpnext := cur_deme^.next;
    dmplast := cur_deme^.last;
    cur_deme^ := tmpdeme;
    cur_deme^.next := dmpnext;
    cur_deme^.last := dmplast;
    count := count+1;

 END;

  WriteLn(' found ', count:5, ' demes in ', prmname);

  WHILE cur_deme^.last <> NIL DO cur_deme := cur_deme^.
   last;

  parsefile(prmname, 'dst');
  WriteLn(' opening distribution file  ', prmname);
  Assign(distfile, prmname);
  {$i-}
  Reset(distfile);
  IF IOResult <> 0 THEN
   BEGIN
    WriteLn(' io error in ', prmname);
    Halt;
   END;

  {$i+}



  cur_freq := NIL;

  WriteLn(' reading ', FileSize(distfile):5,
  ' distributions ');


  loadfreq;

  Close(statefile);

  parsefile(prmname, 'hst');
  Assign(outfile, prmname);
  Rewrite(outfile);


  screenminx := 1;
```

```
  screenminy := 0;
  screenmaxx := xmaxglb;
  screenmaxy := ymaxglb-9;


 END;                                                    {initialize}

Procedure writevert(aline:medstr);
writevert(aline : medstr);
 VAR i, maxlen, topline : Integer;
 BEGIN
  maxlen := 25-WhereY;
  IF Length(aline) <= maxlen THEN
   maxlen := Length(aline);

  FOR i := 1 TO maxlen DO
   WriteLn(aline[i]);
 END;


 PROCEDURE new_page;

 BEGIN
  IF NOT isfirst THEN
   BEGIN
    IF dohard THEN
     BEGIN
      IF isoki THEN
       hardoki(False)            { print hardcopy on okidata
                                                        193}
      ELSE
       hardcopy(False, emode);
      printcount := printcount+1;
      IF (printcount MOD 2) = 0 THEN Write(Lst, Chr(12)
       );

     END;
   END
  ELSE
   isfirst := False;

  ClearScreen;
  defineworld(2, 0.0, 0.0, 100.0, 100.0);

  definewindow(1, screenminx, screenminy, screenmaxx,
```

```
screenmaxy);

setwindowmodeon;

setheaderoff;
setheadertobottom;
defineheader(1,
'  Input theta, output recovered distributions ');

selectworld(2);
selectwindow(1);
drawborder;
setlinestyle(0);
drawline(0.0, 100.0, 100.0, 0.0);
axislabel := 'population of species 2';
i := (25-Length(axislabel)) DIV 2;
IF i < 1 THEN i := 1;
GoToXY(1, i);
writevert(axislabel);
axislabel := 'population of species 1';
i := (80-Length(axislabel)) DIV 2;
IF i < 1 THEN i := 1;
GoToXY(i, 25);
Write(axislabel);
GoToXY(3, 22);
Write('Distribution of distances moved on isocline');
GoToXY(40, 3);
Write('Distribution of theta perturbations');
GoToXY(3, 16); Write('          "cluster"   growth ');
GoToXY(3, 17); Write('           parameter   rate  ');
GoToXY(3, 18); Write('Species 1:', cur_deme^.cval1:4:
1,
'       ', cur_deme^.rate1:4:1);
GoToXY(3, 19); Write('Species 2:', cur_deme^.cval2:4:
1,
'       ', cur_deme^.rate2:4:1);


END;                                              { new_page }

PROCEDURE plotits;
 { plot frequency distributions of p and s type }

VAR
 beta1, beta2, smin, smax, avgval, sumsqs, estsd,
 fcount, xst2, psize
```

```
: Real;
firstzero, lastzero, fnumber, i, j, plotcount,
maxfreq : Integer;
locmax, printcount, timecount : Integer;
done, iszero : Boolean;
xoff, yoff, xmaxval, xmostcom : Real;
strval, timelabel, tmpval : shortstr;
centlabel : medstr;

BEGIN

WHILE cur_freq^.last <> NIL DO cur_freq := cur_freq^.
 last;
WHILE cur_deme^.last <> NIL DO cur_deme := cur_deme^.
 last;


plotcount := 0;
printcount := 0;
ClearScreen;
timecount := 0;


REPEAT                                    { through all sets }
 done := False;


 REPEAT                              { all distributions in set }

  iszero := False;
  j := (timecount*options.listfreq);
  Str(j:7, tmpval);

  timelabel := Concat(' t =', tmpval);


  { write(' now at ',cur_deme^.demeno:5,' ');}

  CASE(plotcount MOD 5) OF
   0 : BEGIN
        xoff := 0.0; yoff := 0.0; psize := 1.0;
       END;
   1 : BEGIN
        xoff := 0.0; yoff := 0.0; psize := 0.5;
       END;
   2 : BEGIN
```

```
      xoff := 0.5; yoff := 0.0; psize := 0.5;
     END;
 3 : BEGIN
      xoff := 0.0; yoff := 0.5; psize := 0.5;
     END;
 4 : BEGIN
      xoff := 0.5; yoff := 0.5; psize := 0.5;
     END;
 END;


WITH cur_freq^ DO
 BEGIN

  { first check for zero frequency array }

  firstzero := 1;
  WHILE (sfreq[firstzero] = 0.0) AND (firstzero <
  maxgen)
  DO firstzero := firstzero+1;
  lastzero := maxgen;
  WHILE (sfreq[lastzero] = 0.0) AND (lastzero > 1)
  DO lastzero := lastzero-1;
  IF lastzero < firstzero THEN iszero := True;

  IF NOT iszero THEN
   BEGIN
    maxfreq := 0;
    FOR i := 1 TO maxgen DO
     IF (Round(sfreq[i]) > maxfreq) THEN
      BEGIN
       maxfreq := Round(sfreq[i]);
       locmax := i;
      END;



    SMIN := xmin+(firstzero*xstep);
    SMAX := xmin+(lastzero*xstep);
    xmaxval := xmin+(xstep*(maxgen+1));
    xmostcom := xmin+(xstep*(locmax+0.5));


    { betaest(beta1, beta2, firstzero, lastzero,
```

```
 sfreq); }

 { now plot distribution }

 display(cur_deme^.demeno, psize, xoff, yoff,
 xmin, xstep,
 maxgen, sfreq, xtitle);
 GoToXY((Trunc((xoff*80)+1)), (Trunc((yoff*24)+1
 )));
 Write(xtitle);
 GoToXY((Trunc((xoff*80)+1)), (Trunc((yoff*24)+(
 psize*20))))
 ;
 Str(xmin:6:2, strval);
 Write(strval);
 GoToXY((Trunc((xoff*80)+(psize*68))), (Trunc((
 yoff*24)+(
 psize*20))));
 Str(xmaxval:6:2, strval);
 Write(strval);
 GoToXY((Trunc((xoff*80)+(psize*34))), (Trunc((
 yoff*24)+2)))
 ;
 Str(xmostcom:6:2, strval);
 centlabel := Concat(strval, timelabel);
 Write(centlabel);

END;                                    { not iszero}


I := 1; avgval := 0.0; fcount := 0.0; sumsqs :=
0.0;
xst2 := (xstep/2.0)+xmin;

WHILE i < maxgen DO
 BEGIN
  avgval := avgval+(sfreq[i]*((i*xstep)+xst2));
  sumsqs := sumsqs+(sfreq[i]*(Sqr((i*xstep)+xst2)
  ));
  fcount := fcount+sfreq[i];
  i := i+1;
 END;

{ writeln(' sum values ',avgval:10:3,' sum of
squares ', sumsqs:10:3);}
IF fcount > 0 THEN
```

```
   avgval := avgval/fcount
  ELSE
   avgval := 0.0;

  IF fcount > 1 THEN
   BEGIN
    estsd := fcount*Sqr(avgval);
    estsd := sumsqs-estsd;
    estsd := Sqrt(estsd/(fcount-1));
   END
  ELSE
   estsd := 0.0;


  { writeln(' average value ',avgval:10:3,'
  standard deviation ', estsd:10:3);}

 END;                                          { with cur_freq}

WriteLn(outfile, ' Mean value ', avgval:10:3,
' standard deviation ',
estsd:10:3);
WriteLn(outfile, ' model type ', options.stype:2,
' perturb type ',
options.ptype:2);

WriteLn(outfile);

IF ((plotcount MOD 5) = 0) OR ((plotcount MOD 5) =
4) THEN

 IF NOT dohard THEN
  BEGIN
   GoToXY(35, 25); Write('Press return to continue'
   );
   ReadLn;
   ClearScreen;
  END
 ELSE
  BEGIN
   IF isoki THEN
    hardoki(False)          { print hardcopy on okidata
                                                  193}
   ELSE
    hardcopy(False, emode);
   printcount := printcount+1;
```

```
      IF (printcount MOD 2) = 0 THEN Write(Lst, Chr(12
       ));
      ClearScreen;
     END;


    plotcount := plotcount+1;

   IF cur_freq^.next <> NIL THEN
    cur_freq := cur_freq^.next
   ELSE
    done := True;


  UNTIL done;

  loadfreq;

  timecount := timecount+1;

 UNTIL EoF(distfile);


 IF NOT dohard THEN
  BEGIN
   GoToXY(35, 25);
   Write(' Press return to continue '); ReadLn;
  END;



END;                                    { plotits procedure }


PROCEDURE plotitB;

VAR
 beta1, beta2, smin, smax, avgtheta, avgdist, fcount,
 xst2 : Real;
 firstzero, lastzero, fnumber, i, j, curwindow :
 Integer;
 done : Boolean;
```

```
BEGIN

 WHILE cur_freq^.last <> NIL DO cur_freq := cur_freq^.
  last;
 WHILE cur_deme^.last <> NIL DO cur_deme := cur_deme^.
  last;
 done := False;

 REPEAT

  { write(' now at ',cur_deme^.demeno:5,' ');}

  curwindow := ((cur_deme^.demeno-1) MOD 4);

  IF (curwindow = 0) AND (NOT dohard) AND (NOT isfirst
  ) THEN
   BEGIN
    GoToXY(35, 25);
    Write(' Press return to continue '); ReadLn;
   END;


  IF curwindow = 0 THEN new_page;


  IF options.ptype <> 'R' THEN
   BEGIN
    WITH cur_freq^ DO
     BEGIN

      { first calculate beta values }

      firstzero := 1;
      WHILE (sfreq[firstzero] = 0.0) AND (firstzero <
      maxgen)
      DO firstzero := firstzero+1;

      lastzero := maxgen;

      WHILE (sfreq[lastzero] = 0.0) AND (lastzero > 0)
      DO lastzero := lastzero-1;

      IF lastzero < firstzero THEN
```

```
 BEGIN
  WriteLn('Error -- Zero frequency array read ')
  ;
  Halt;
 END;


SMIN := xmin+(firstzero*xstep);
SMAX := xmin+(lastzero*xstep);


betaest(beta1, beta2, firstzero, lastzero, sfreq
);

{ now plot theta }

display(curwindow, 0.19, ((curwindow+1)/5), ((
curwindow)/5),
xmin, xstep, maxgen, sfreq, xtitle);

{ display(thiswindow:integer;scale,xoff,yoff,
xmin,xstep:real;
maxgen:integer;x:vector;xtitle:title);}



I := 1; avgtheta := 0.0; fcount := 0.0;
xst2 := (xstep/2.0)+xmin;

WHILE i < maxgen DO
 BEGIN
  avgtheta := avgtheta+(sfreq[i]*((i*xstep)+xst2
  ));
  fcount := fcount+sfreq[i];
  i := i+1;
 END;
avgtheta := avgtheta/fcount;


 { write(' average theta ',avgtheta:10:3); }

END;                                    { with cur_freq}

WITH cur_deme^ DO
 BEGIN
  WriteLn(outfile, '  STATISTICS FOR DEME NUMBER '
```

```
    , demeno:5);
    WriteLn(outfile, ' first population ':30,
    ' Mean theta ',
    avgtheta
    :10:3);
    WriteLn(outfile, ' c - ', cval1:5:1, ' r - ',
             rate1:5:1,' final population ',pop1:5:1,
            ' model type ', options.stype:2,
            ' perturb type ',
    options.ptype:2);
    WriteLn(outfile, ' second population ');
    WriteLn(outfile, ' c - ', cval2:5:1, ' r - ',
    rate2:5:1,
    ' final population ',
    pop2:5:1, ' first beta ', beta1:6:3,
    ' second beta ',
    beta2:6:3);

    WriteLn(outfile);

   END;                                { with cur_deme block }

 END;                                 { if option.ptype<>'r' }


{  now do distance distribution }

IF cur_freq^.next <> NIL THEN
 cur_freq := cur_freq^.next
ELSE
 done := True;

IF NOT done THEN
 BEGIN

  WITH cur_freq^ DO
   BEGIN

    { first calculate beta values }

    firstzero := 1;
    WHILE (sfreq[firstzero] = 0.0) AND (firstzero <
    maxgen)
    DO firstzero := firstzero+1;

    lastzero := maxgen;
```

```
WHILE (sfreq[lastzero] = 0.0) AND (lastzero > 0)
DO lastzero := lastzero-1;

IF lastzero < firstzero THEN
BEGIN
  WriteLn('Error -- Zero frequency array read ')
  ;
  Halt;
 END;


SMIN := xmin+(firstzero*xstep);
SMAX := xmin+(lastzero*xstep);


betaest(beta1, beta2, firstzero, lastzero, sfreq
);

{ now plot theta }

display(curwindow, 0.19, ((curwindow)/5), ((
curwindow+1)/5),
xmin, xstep, maxgen, sfreq, xtitle);

I := 1; avgdist := 0.0; fcount := 0.0;
xst2 := (xstep/2.0)+xmin;

WHILE i < maxgen DO
 BEGIN
  avgdist := avgdist+(sfreq[i]*((i*xstep)+xst2))
  ;
  fcount := fcount+sfreq[i];
  i := i+1;
 END;
avgdist := avgdist/fcount;


 { writeln(' average distance ',avgdist:10:3);}

END;                                    { with cur_freq}

WITH cur_deme^ DO
 BEGIN
  WriteLn(outfile, '  STATISTICS FOR DEME NUMBER '
  , demeno:5);
```

```
      WriteLn(outfile, ' first population ':30,
      ' mean distance ',
      avgdist:10:3);
      WriteLn(outfile, ' c - ', cval1:5:1, ' r - ',
      rate1:5:1,' final population ',
      pop1:5:1, ' model type ', options.stype:2,
      ' perturb type ',
      options.ptype:2);
      WriteLn(outfile, ' second population ');
      WriteLn(outfile, ' c - ', cval2:5:1, ' r - ',
      rate2:5:1,
      ' final population ',
      pop2:5:1, ' first beta ', beta1:6:3,
      ' second beta ',
      beta2:6:3);

      WriteLn(outfile);
     END;                                { with cur_deme block }

   END;                                        { not done yet }


   IF cur_freq^.next <> NIL THEN
    cur_freq := cur_freq^.next
   ELSE
    done := True;

   IF cur_deme^.next <> NIL THEN
    cur_deme := cur_deme^.next
   ELSE
    done := True;

  UNTIL done;

  IF NOT dohard THEN
   BEGIN
    GoToXY(35, 25);
    Write(' Press return to continue '); ReadLn;
   END;


END;                                          { plotitb procedure }

PROCEDURE dumpworld;
BEGIN
```

```
 copyscreen;
 leavegraphic;

 WriteLn(' window mode is ', windowmode);
 WriteLn(' Global world  parameters  x1, y1, x2, y2 ');
 WriteLn(x1wldglb:10:2, y1wldglb:10:2, x2wldglb:10:2,
 y2wldglb:10:2);
 WriteLn(
 ' Global translation paramters Axglb,Bxglb,Ayglb,Byglb');
 WriteLn(Axglb:10:2, Bxglb:10:2, Ayglb:10:2, Byglb:10:2);
 WriteLn('   Global parameters x1,y1,x2,y2 ');
 WriteLn(x1glb:10, y1glb:10, x2glb:10, y2glb:10);
 WriteLn(' contents of world 2 ');
 WITH world[2] DO
  WriteLn(x1:10:2, y1:10:2, x2:10:2, y2:10:2);

 Write(' press return to continue '); ReadLn;

 entergraphic;
 selectscreen(2);
 copyscreen;
 selectscreen(1);


END;

BEGIN                                          { main program }

 initgraphic;
 leavegraphic;
 setbreakoff;

 ClrScr;

 Write(' enter base name of data files ');
 ReadLn(prmname);
 WriteLn(
 ' Do you want expanded view of each histogram? ');
 Write(' Enter Y if you do or any other key if not ');
 ReadLn(anschr);
 WriteLn;
 IF anschr IN ['y', 'Y'] THEN
  doblowup := True
 ELSE
  doblowup := False;
```

```
WriteLn(' Do you want a printout of each plot? ');
Write(' Enter Y if you do or any other key if not ');
ReadLn(anschr);
WriteLn;
IF anschr IN ['y', 'Y'] THEN
 BEGIN
  dohard := True;
  WriteLn(
   ' Do you have an Okidata 193 or an Epson (IBM)
       compatible printer?'    );
  REPEAT
   Write(' Enter O if Okidata or E if Epson '); ReadLn
   (anschr);
   WriteLn;
  UNTIL anschr IN ['O', 'o', 'E', 'e'];

  IF anschr IN ['O', 'o'] THEN isoki := True
  ELSE BEGIN
   isoki := False;
   Write(' Enter H for half size print '); ReadLn(
   anschr);
   WriteLn;
   IF anschr IN ['H', 'h'] THEN
    emode := 1
   ELSE
    emode := 0;
  END;
 END
ELSE
 dohard := False;

isfirst := True;
printcount := 0;

initialize;
{ writeln(' memory left ',memavail);}

entergraphic;
{ new_page;}

CASE options.stype OF
 'B' : plotitb;
 'S', 'P' : plotits;
END;
IF dohard THEN
 BEGIN
```

```
   IF isoki THEN
    hardoki(False)
   ELSE
    hardcopy(False, emode);
   Write(Lst, Chr(12));
  END;

 leavegraphic;
 Close(distfile);
 Close(outfile);


END.
```