Ecological Bulletins 48: 191–203. Copenhagen 2000

Structure of the VORTEX simulation model for population viability analysis

Robert C. Lacy

Lacy, R. C. 2000. Structure of the VORTEX simulation model for population viability analysis. – Ecol. Bull. 48: 191–203.

The structure of the VORTEX computer simulation model for population viability analysis is outlined. The program flow is described here in order to provide a detailed specification of the structure of a widely used population viability analysis model. VORTEX is an individual-based simulation program that models the effects of mean demographic rates, demographic stochasticity, environmental variation in demographic rates, catastrophes, inbreeding depression, harvest and supplementation, and metapopulation structure on the viability of wildlife populations. The model facilitates analysis of density-dependent reproduction and changing habitat availability, and most demographic rates can optionally be specified as flexible functions of density, time, population gene diversity, inbreeding, age, and sex. VORTEX projects changes in population size, age and sex structure, and genetic variation, as well as estimating probabilities and times to extinction and recolonization.

R. C. Lacy (rlacy@ix.netcom.com), Dept of Conservation Biology, Daniel F. & Ada L. Rice Center, Chicago Zoological Society, Brookfield, IL 60513, USA.

"VORTEX and like programs do exactly what they are told to do, as constrained by the static single-species models that provide their structure. They can be useful for various purposes so long as the user understands what the programs are doing ..." (Caughley and Gunn 1996, p. 208).

The complexity and multiplicity of processes influencing the dynamics of natural populations of animals and plants means that population viability analysis (PVA) models are also frequently complex. Different models incorporate different population processes. Individual population processes can be modeled in various ways, requiring different sets of driving variables, using different equations to define the processes, and providing different output to describe the population dynamics. Users of PVA models should understand the basic structure of the models they use, and it is important that models used for scientific studies and conservation efforts can be examined and replicated. Yet often the details of PVA computer programs are not available to the users, because the code is proprietary information or otherwise not provided to users, or simply because the task of reading and understanding the source code for large and complex programs is formidable.

One possible remedy to the problem of PVA users needing to understand the models being used is for practitioners to develop their own computer programs. This would result in the user having a full understanding of a model that would be specifically designed for the analysis. Development of user-specific and case-specific models is usually not practical, however, as many population biologists are not skilled computer programmers, and the time required to develop a complex model is often prohibitive. Moreover, a complex computer program developed by and used by one person will sometimes contain serious programming errors. The testing of programs that are widely used may be a necessary prerequisite for reliable population viability analyses to be employed effectively in biodi-

versity conservation. Finally, the flexibility and expansive capabilities of generic PVA software to model a large diversity of population processes will often lead PVA practitioners to consider threats to population viability that would otherwise have been neglected.

Widely available PVA software can serve the same role as do statistical analysis packages. The ease of use, flexible application to diverse needs, and extensive prior testing facilitate many applications that would not otherwise be attempted. Ideally, perhaps, all users of statistical methods would write their own programs or otherwise study the code of the software entrusted for the analyses. More practically, confidence is gained in the reliability of generic software tools as more people use the programs and compare the generated results to expectations from statistical theory and to results for simple and well known cases. Also, users of statistical software are expected to be sufficiently familiar with the methods of statistical analysis to be able to choose appropriate models to apply to their problem, to be able to provide the proper input, and to be able to interpret the results.

Unlike the situation for statistical methods, however, there are not yet widely accepted and published accounts of standard methods for population viability analysis. The methods of population-based models (e.g., Starfield and Bleloch 1986, Burgman et al. 1993) are extensions of the methods of population ecology and demography (e.g., Pielou 1977, Caswell 1989), but many details of model construction require decisions about algorithms and methods that are not fully delineated in general treatments. The methods of individual-based PVA models have been only cursorily described in the scientific literature. Below is an outline of one widely used PVA software package, VORTEX, ver. 8.20. The basic approach taken in the VORTEX model is described in Lacy (1993), in Lindenmayer et al. (2000) and other papers describing applications of VORTEX, and in the software manual (Miller and Lacy 1999). Detailed documentation of the program flow and algorithms is provided here, so that users of VORTEX can confirm that the model is performing the analyses that are intended, and so that PVA practitioners in general can see an example of the structure of an individual-based PVA model. The VORTEX program is available at http://www2.netcom.com/~rlacy/vortex.html.

The pseudo-code presented below is an English-language outline of the program flow and primary algorithms used by VORTEX (which is written in the C programming language). This pseudo-code omits coding for: 1) input routines for reading parameters from files and/or keyboard; 2) output routines for writing results to files; 3) specification of default parameter values; 4) checks for illegal values, error handling; 5) memory management and initialization of memory; 6) details of C coding to achieve algorithms; 7) routines for on-line help; 8) routines for graphical display of functions specifying demographic rates, population sizes during simulations, simulation results; 9) routines for evaluating equations that specify demographic rates (e.g., breeding, mortality) as functions of population and individual variables (e.g., population size, gene diversity, year, age, sex, inbreeding) (see note 3 below); 10) tallies of mean within-population statistics and metapopulation summaries; 11) algorithms for calculating basic statistics, such as means, standard deviations, standard errors, and medians across years and across iterations.

Variables for storing input, intermediate calculations, and output are indicated in the pseudo-code by italicized labels. Many of the variables are arrays (e.g., a value stored for each population, or for each age class, or for each individual), as suggested by the loops within which they are calculated and used. The indices of such arrays are indicated within brackets (e.g., *MortalityRate[p][s][x]*, for each population, *p*, sex, *s*, and age, *x*). VORTEX uses many more variables (not shown in the pseudo-code) for facilitating calculations and accumulating sums, sums of squares, and other components needed for the basic statistics reported in the output.

In the pseudo-code, loops are indicated with FOR: and END LOOP statements, or by WHILE: and END WHILE statements. Conditional actions are indicated by IF: and END IF statements, or by IF:, ELSE:, and END IF/ELSE statements. BREAK indicates that program flow exits from the bottom of a loop. CONTINUE indicates that program flow jumps back to the next value at the top of the loop. Multiplication is indicated by the asterisk (*) symbol; ^ indicates exponentiation; SQRT indicates the positive square root.

Function modules defined outside of the main body of the pseudo-code program are labeled in the form **FUNCTION()**, and are specified below the main **VORTEX()** program. The actual C code is subdivided into many smaller functions; the pseudo-code shows only the flow of the overall program and its largest modules. The functions **RAND()** and **NRAND()** indicate, respectively, that a random number is generated from the uniform 0-1 distribution or from a unit normal distribution.

Explanatory comments, following pseudo-code sections, are preceded by //. More extensive explanations are given in notes following the code.

As an individual-based PVA simulation model, VORTEX represents each individual in memory, simulates life events (such as sex determination, breeding, mortality, and dispersal) which could occur to each individual, and monitors the status of each individual and the population as a whole. The characteristics tracked for each animal are: sex, alive/dead status, population membership, age, inbreeding coefficient, and two alleles at each of six loci. In addition, VORTEX maintains a matrix of kinship coefficients between all pairs of living animals, as this provides inbreeding coefficients for any offspring.

VORTEX models changes to a population as a series of discrete events that occur once per year (or other time interval). The annual sequence of demographic events is:

breeding; mortality; age 1 yr; migrate (disperse) among populations; harvest (managed removals); supplementation (managed additions); carrying capacity truncation; census (Fig. 1). Occurrences of events are probabilistic; demographic stochasticity emerges from chance variation in which individuals breed, die, and are of each sex. Environmental variation in demographic rates is imposed by sampling rates from specified distributions during each simulated year. Catastrophes, which occur with specified probabilities, cause one-year reductions in reproduction and survival. Genetic effects are modeled as reduced survivorship of inbred individuals.

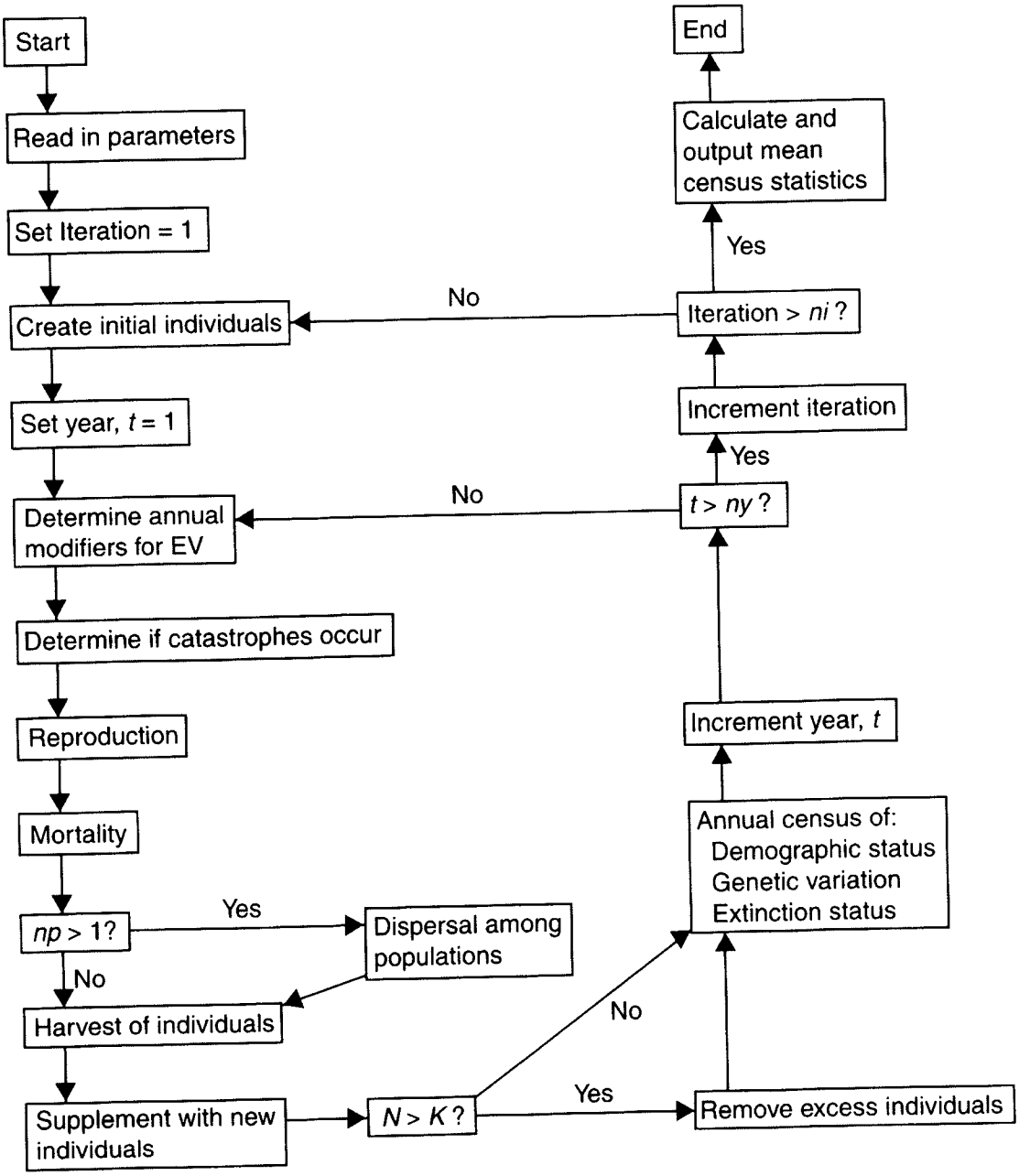

Fig. 1. Flow chart of the primary components of the Vortex simulation. Each step from "Create initial individuals" through "Annual census" is applied to each population in a modeled metapopulation. t = year; ny = number of years simulated; np = number of populations; ni = number of iterations; N = population size; K = carrying capacity; EV = environmental variation.

VORTEX program pseudo-code

```
BEGIN PROGRAM VORTEX():
Initialize random number generator; // See Note 1.
  FOR (each scenario):
    READ_SPECIES_PARAMETERS();
    IF (NumberOfPopulations > 1):
      READ_MIGRATION_PARAMETERS();
// VORTEX describes dispersal between populations as "migration."
    END IF
    FOR (each population, p):
      READ_POPULATION_PARAMETERS(p);
    END (population) LOOP
    FOR (each population, pSource):
// Calculate cumulative migration rates for each pairwise transition between populations.
        Set CumMigrationProb[pSource][1] = MigrationProb[pSource][1];
        FOR (each destination population, pDestination, greater than 1):
          Set CumMigrationProb[pSource][pDestination] = CumMigrationProb[pSource][pDestination -1]+ MigrationProb[pSource][pDestination];
        END (pDestination) LOOP
    END (pSource) LOOP
    Set NumberLethals = InbreedingGeneticLoad * ProportionLoadDueToLethals;
    Set LethalEquivalents = InbreedingGeneticLoad * (1 - ProportionLoadDueToLethals);
// See Note 2.
    FOR (each population, p):
      Set GlobalBreedEV[p] = BreedEV[p] * EVConcordanceAmongPopulations;
      Set LocalBreedEV[p] = SQRT( BreedEV[p] ^2 - GlobalBreedEV[p] ^2);
// Partition Environmental Variation in breeding (BreedEV) into the component that is common to all populations (GlobalBreedEV) and the component that is specific to each population. (LocalBreedEV); TotalEV ^ 2 = GlobalEV ^ 2 + LocalEV ^2
// Note: EVs are given as standard deviations.
      FOR (each sex, s):
        FOR (each age, x, up to age of breeding):
          Set GlobalMortEV[p][s][x] = MortEV[p][s][x] * EVConcordanceAmongPopulations;
          Set LocalMortEV[p][s][x] = SQRT(MortEV[p][s][x] ^2 - GlobalMortEV[p][s][x] ^2);
// Partition EV in mortality (MortEV) into the component that is common to all populations (GlobalMortEV) and the component that is specific to each population (LocalMortEV).
        END (age) LOOP
      END (sex) LOOP
      Set GlobalKEV[p] = KEV[p] * EVConcordanceAmongPopulations;
      Set LocalKEV[p] = SQRT( KEV[p] ^2 - GlobalKEV[p] ^2);
// Partition EV in carrying capacity (KEV) into the component that is common to all populations (GlobalKEV) and the component that is specific to each population (LocalKEV).
    END (population) LOOP
    FOR (each iteration):
      FOR (each population):
        Create initial individuals, assigning population, sex, age, alive/dead status, inbreeding coefficient and kinships (initially 0), and alleles at six loci;
        FOR (each of five non-neutral loci, l):
          FOR (each founder allele, a):
// The probability that a given founder allele is a lethal is NumberLethals / 10, because there are 10 alleles across the five diploid loci.
            IF (RAND() < NumberLethals / 10):
              Set Lethal[l][a] = TRUE;
// Allele, a, of locus, l, is a recessive lethal.
            ELSE:
              Set Lethal[l][a] = FALSE;
            END IF/ELSE
          END (founder allele) LOOP
        END (locus) LOOP
        Display initial population sizes on screen and write to output files;
      END (population) LOOP
      FOR (each year):
        IF (NumberPopulations > 1):
          GLOBAL_EV_RANDS();
// Generate random numbers for specifying environmental variation, concordant across populations, for the year.
        END (NumberPopulations) IF
        FOR (each population, p):
          LOCAL_EV_RANDS();
          CATASTROPHES(p);
// Determine if catastrophes occur that year.
          Determine carrying Capacity (K) for year;
// See Note 3.
          Add LocalKEVNRand * LocalKEV[p] to CarryingCapacity[p];
// Adjust K for local EV.
          Add GlobalKEVNRand * GlobalKEV[p] to CarryingCapacity[p];
// Adjust K for global EV.
          BREED(p);
// Go through breeding cycle to produce offspring.
          MORTALITY(p);
// Determine who dies that year.
        END (population) LOOP
        Add 1 to the age of each animal;
        IF (NumberPopulations > 1):
          MIGRATE();
```

```
// Determine which animals migrate between populations.
            END IF
            FOR (each population, p):
               IF (year during which animals are to be
               harvested):
                  HARVEST(p);
               END (harvest year) IF
            END (population) LOOP
            FOR (each population, p):
               IF (year during which animals are to be
               supplemented):
                  SUPPLEMENT(p);
               END (supplement year) IF
            END (population) LOOP
            FOR (each population, p):
               Tally PopulationSize[p];
               IF (population is not extinct AND
               population was not extinct prior year):
// "Extinction" can be defined by the user as the absence of
one sex, or as the population size falling below a specified
lower limit.
                  r[p] = log(PopulationSize[p] /
                  PopulationSizePriorYear[p]);
// Calculate population growth rate ( r ).
               END IF
               IF (not extinct AND PopulationSize[p], N
               > CarryingCapacity[p], K):
                  FOR (each living animal):
                     IF (RAND() > K / N):
// Stochastically kill excess above K.
                        Animal dies;
                     END IF
                  END (each animal) LOOP
               END (N > K) IF
               Tally PopulationSize[p];
               IF (population is extinct):
                  Decrement NumberExtantPopulations[p];
                  IF (population was not extinct in prior
                  year):
                     Set YearExtinct[p] = CurrentYear;
                     IF (population has not been
                     recolonized):
// First extinction
                        Set TimeToExtinction[p] =
                        CurrentYear;
                        Increment
                        NumberOfExtinctions[p];
                     ELSE:
// Re-extinction of population
                        Set TimeToReextinction[p] =
                        CurrentYear –
                        YearOfRecolonization[p];
                     END (recolonized) IF/ELSE
                  END (was not extinct) IF
               ELSE:
// Not extinct
                  IF (population was extinct in prior year):
// Recolonization
                     Set YearRecolonized[p] = CurrentYear;
                     Set TimeToRecolonization[p] =
                     CurrentYear – YearExtinct[p];
                     Increment
                     NumberOfRecolonizations[p];
                  END (was extinct) IF
                  Set YearExtinct[p] = 0;
// Flag for not extinct
               END (extinct) IF/ELSE
               Display PopulationSize[p] on screen graph;
            END (population) LOOP
            FOR (each population, p):
               CALC_GENETIC_METRICS(p);
            END (population) LOOP
         END (year) LOOP
      END (iteration) LOOP
// At this point, the simulation is complete and summary
statistics can be calculated.
      FOR (each population, p):
         Calculate and report means, SDs, and SEs across
         iterations for
            Population growth rate: r[p] = N[CurrentYear]
            / N[PreviousYear]
            TimeToExtinction[p]
            TimeToRecolonization[p]
            TimeToReextinction[p]
         FOR (each year):
            Calculate and output means, SDs, and SEs
            across iterations for:
               Probability of extinction, PE[p]
// SE = SQRT[PE * (1 - PE) / NumberIterations]
               PopulationSize[p]
               GeneDiversity[p]
// Gene Diversity = Heterozygosity expected under Hardy-
Weinberg equilibrium
               ObservedHeterozygosity[p]
// = 1 - mean inbreeding coefficient
               NumberAlleles[p]
               LethalFrequency[p]
         END (year) LOOP
      END (population) LOOP
      Calculate and report within-population means of
      above summary statistics;
      Call program for displaying graphical displays of
      trends in:
         PopulationSize
         GeneDiversity
         Mean inbreeding coefficient (1 -
         ObservedHeterozygosity)
         Probability of population persistence to year
         Probability of extinction in that time interval
      Read in DoAnotherScenario?
      IF (DoAnotherScenario? is FALSE):
         BREAK from scenario LOOP
```

```
    END IF
  END (scenario) LOOP
END PROGRAM VORTEX()

BEGIN FUNCTION
READ_SPECIES_PARAMETERS():
// Get input parameters from keyboard or input file, describing simulation parameters, inbreeding effects, and basic species life history.
  Read in Input/Output file names;
  Read in NumberOfIterations;
  Read in NumberOfYears;
  Read in ExtinctionDefinition;
// Extinction can be defined as no animals of one sex, or as the population size falling below a specified minimum.
  Read in NumberOfPopulations;
  Read in InbreedingGeneticLoad;
  Read in ProportionLoadDueToLethals;
  Read in EVCorrelationBetweenReproductionAndSurvival?;
  IF (NumberOfPopulations > 1):
    Read in EVConcordanceAmongPoulations;
  END IF
  Read in NumberTypesOfCatastrophes;
  Read in Monogamous/Polygynous/Hermaphroditic?;
  Read in FemaleBreedingAge;
  Read in MaleBreedingAge;
  Read in MaximumAge;
  Read in SexRatio at birth;
  Read in MaximumLitterSize;
  Read in DensityDependentBreeding?
// The pseudocode for modeling density dependent breeding is not given below.
END FUNCTION READ_SPECIES_PARAMETERS()

BEGIN FUNCTION
READ_MIGRATION_PARAMETERS():
// Get input population structure and migration patterns.
  Read in MigrationAges;
  Read in MigrationSexes;
  Read in MigrationSurvival;
  Read in MigrationDensity;
  FOR (each population, pSource):
    FOR(each other population, pDestination):
      Read in MigrationProb[pSource][pDestination];
    END LOOP
  END LOOP
END FUNCTION
READ_MIGRATION_PARAMETERS()

BEGIN FUNCTION
READ_POPULATION_PARAMETERS(for population p):
  Read in ProportionFemalesBreeding[p];
  Read in BreedEV[p];
// Environmental variation is specified as a standard deviation.
  Read in Litter size distribution (either as MeanLitterSize[p] and SDLitterSize[p] or as the fully specified distribution of ProbLitterSize[p][n]);
  FOR (each age, x, up to FemaleBreedingAge ):
    Read in FemaleMortality[p][x];
    Read in FemaleMortalityEV[p][x];
  END (age) LOOP
  FOR (each age, x, up to MaleBreedingAge ):
    Read in MaleMortality[p][x];
    Read in MaleMortalityEV[p][x];
  END (age) LOOP
  FOR (each type of catastrophe, c):
    IF (NumberPopulations > 1):
      Read in GlobalOrLocal[p][c];
    END IF
    Read in CatastropheFrequency[p][c];
    Read in CatastropheBreedSeverity[p][c];
    Read in CatastropheSurvivalSeverity[p][c];
  END (catastrophe) LOOP
  CALC_DETERMINISTIC_GROWTH(p);
// Calculate deterministic population growth rate, generation time, and stable age distribution from mean birth and death rates. Effects of any catastrophes are averaged across years
  Read in ProportionMalesInBreedingPool[p];
// See Note 4.
  IF (initial numbers of animals are to be distributed according to the stable age distribution):
    Determine initial numbers of animals in each age-sex class;
// The stable age distribution would rarely assign whole numbers to each age-sex class. Integral numbers are assigned that most closely match the desired distribution.
  ELSE (does not start at stable age distribution):
    FOR (each sex):
      FOR (each age up to MaximumAge):
        Read in initial number of animals;
      END LOOP
    END LOOP
  END (stable age distribution) IF/ELSE
  Read in CarryingCapacity[p] (K);
// K may be specified as a function of year or other parameters. See Note 3.
  Read in KEV[p];
  Read in Harvest[p]?;
  IF (Harvest[p]? = Yes):
    Read in FirstYearHarvest[p], LastYearHarvest[p], HarvestInterval[p];
    FOR (each age, x, up to FemaleBreedingAge):
// For harvest, all adults are treated in the same age category.
      Read in NumberFemalesToBeHarvested[p][x];
    END LOOP
    FOR (each age, x, up to MaleBreedingAge):
      Read in NumberMalesToBeHarvested[p][x];
    END LOOP
```

```
  END IF
  Read in Supplement[p]?;
  IF (Supplement[p]? = Yes):
     Read in FirstYearSupplementation[p],
     LastYearSupplementation[p],
     SupplementationInterval[p];
     FOR (each age, x, up to FemaleBreedingAge):
        Read in NumberFemalesToBeSupplemented[p][x];
     END LOOP
     FOR (each age, x, up to MaleBreedingAge):
        Read in NumberMalesToBeSupplemented[p][x];
     END LOOP
  END IF
END FUNCTION
READ_POPULATION_PARAMETERS()

BEGIN FUNCTION
CALC_DETERMINISTIC_GROWTH(for population p):
// Use standard life table analysis; solve the Euler equation to find the deterministic growth rate.
   Set fecundity, M = MeanLitterSize[p] * (1 - SexRatio);
// SexRatio is proportion males at birth.
   FOR (each type of catastrophe, c):
// Adjust M for catastrophes.
      Multiply M by CatastropheFrequency[p][c] *
      CatastropheBreedSeverity[p][c] + (1 -
      CatastopheFrequency[p][c]);
   END (catastrophe) LOOP
   FOR (each age, x):
      Set female survival, P[x] = 1 - FemaleMortality[p][x];
      FOR (each type of catastrophe, c):
// Adjust P[x] for catastrophes.
         Multiply P[x] by CatastropheFrequency[p][c] *
         CatastropheSurvivalSeverity[p][c] + (1 -
         CatastopheFrequency[p][c]);
      END (catastrophe) LOOP
      Multiply cumulative survivorship, L[x], by P[x];
      IF (x >= FemaleBreedingAge):
         Add L[x] * M to SumLxMx;
         Add x * L[x] * M to SumAgeLxMx;
      END IF
   END (age) LOOP
   Set R0 = SumLxMx;
   Set GenerationTime[p] = SumAgeLxMx / SumLxMx;
// Preliminary estimate
   Set Lambda = R0 ^ (1/GenerationTime);
// Preliminary estimate
   Solve Euler equation by iterative approximation, to yield precise Lambda;
   Set r[p] = log(Lambda);
   Set GenerationTime[p] = log(R0) / r[p];
   FOR (each age, x):
// Determine stable age distribution.
      Set StableAgeClassSize[p][x] = (1 - SexRatio) * L[x] /
      (Lambda ^ x);
      Add StableAgeClassSize[p][x] to
      SumStableAgeClassSize[p];
   END LOOP
// Repeat age distribution calculations for males, but use female-based Mx and Lambda.
   FOR (each age, x):
      Set male survival, P[x] = 1 - MaleMortality[p][x];
      FOR (each type of catastrophe, c):
// Adjust P[x] for catastrophes.
         Multiply P[x] by CatastropheFrequency[p][c] *
            CatastropheSurvivalSeverity[p][c] + (1 -
            CatastopheFrequency[p][c]);
      END (catastrophe) LOOP
      Multiply cumulative survivorship, L[x], by P[x];
   END LOOP
   FOR (each age, x):
// Determine stable age distribution.
      Set StableAgeClassSize[p][x] = SexRatio * L[x] /
      (Lambda ^ x);
      Add StableAgeClassSize[p][x] to
      SumStableAgeClassSize[p];
   END LOOP
   FOR (each age, x, and sex, s):
      Divide StableAgeClassSize[p][s][x] by
      SumStableAgeClassSize[p];
   END LOOP
END FUNCTION
CALC_DETERMINISTIC_GROWTH()

BEGIN FUNCTION GLOBAL_EV_RANDS():
   FOR (each type of catastrophe):
      Set GlobalCatastropheRand = RAND();
// Select random number to determine if global catastrophe occur. See Note 5.
   END (catastrophe) LOOP
   Set GlobalBreedEVRand = RAND();
// Select random number for specifying EV in breeding for that year.
   Set GlobalBreedEVNRand = NRAND();
// Select random normal deviate for specifying EV in breeding for year. Whether EVRand or EVNRand will be used depends on the magnitude of EV. See Note 6.
   Set GlobalBreedEVNRand to same sign as
   GlobalBreedEVRand;
   IF (EVCorrelationBetweenReproductionAndSurvival? =
   No):
      Set GlobalMortEVRand = RAND();
// Select random 0-1 number for specifying EV mortality for that year.
      Set GlobalMortEVNRand = NRAND();
// Select random normal deviate for specifying EV in mortality for that year. Whether EVRand or EVNRand will be used depends on the magnitude of EV.
      Set GlobalMortEVNRand to same sign as
      GlobalMortEVRand;
      Set GlobalKEVNRand = NRAND();
```

```
// Select random normal deviate for specifying EV in K for year.
   ELSE (EV in breeding is correlated with EV in mortality):
      Set GlobalMortEVRand = GlobalBreedEVRand;
      Set GlobalMortEVNRand = GlobalBreedEVNRand;
      Set GlobalKEVNRand = GlobalBreedEVNRand;
   END (EV correlation) IF/ELSE
END FUNCTION GLOBAL_EV_RANDS()

BEGIN FUNCTION LOCAL_EV_RANDS():
   Set LocalBreedEVRand = RAND();
// Select random number for specifying EV in breeding for year.
   Set LocalBreedEVNRand = NRAND();
// Select a random normal deviate for specifying EV in breeding.
   Set LocalBreedEVNRand to same sign as
   LocalBreedEVRand;
   IF (EVCorrelationBetweenReproductionAndSurvival? =
   FALSE):
      Set LocalMortEVRand = RAND();
// Select random number for specifying EV in mortality.
Set LocalMortEVNRand = NRAND();
// Select random normal deviate for specifying EV in mortality.
      Set LocalMortEVNRand to same sign as
      LocalMortEVRand;
      Set LocalKEVNRand = NRAND();
// Select random normal deviate for specifying EV in K.
   ELSE (EV in breeding is correlated with EV in mortality):
      Set LocalMortEVRand = LocalBreedEVRand;
      Set LocalMortEVNRand = LocalBreedEVNRand;
      Set LocalKEVNRand = LocalBreedEVNRand;
   END (EV correlation) IF/ELSE
END FUNCTION LOCAL_EV_RANDS()

BEGIN FUNCTION CATASTROPHES(for population p):
   FOR (each type of catastrophe, c):
      IF (Catastrophe is local in effect):
         IF (RAND() < CatastropheFrequency[p][c]):
// See Note 5.
            Set CatastropheFlag[c] = TRUE;
// Catastrophe has occurred.
         ELSE:
            Set CatastropheFlag[c] = FALSE;
         END (catastrophe) IF/ELSE
      ELSE:
         IF (GlobalCatastropheRand <
         CatastropheFrequency[p][c]):
            Set CatastropheFlag[c] = TRUE;
         ELSE:
            Set CatastropheFlag[c] = FALSE;
         END (catastrophe) IF/ELSE
      END (Local/Global catastrophe) IF/ELSE
   END (catastrophe) LOOP
END FUNCTION CATASTROPHES()

BEGIN FUNCTION BREED(for population p):
// Find breeders for the year ...
   FOR (each living animal in the population):
      IF (sex = female AND age >= FemaleBreedingAge):
         Add female to breeding pool;
      END IF
      IF (not hermaphroditic):
         IF (sex = male AND age >= MaleBreedingAge):
            IF (RAND() <
            ProportionMalesInBreedingPool[p]):
               Add male to breeding pool;
            END IF
         END IF
      END IF
   END (animal) LOOP
   IF (no males selected for breeding pool, but adult males do exist):
      Add one male at random to breeding pool;
   END IF
   IF (monogamous):
      FOR (each male in breeding pool, m):
         Set MaleUsed[m] = FALSE;
// Flag to indicate male is available for pairing
      END LOOP
   END IF
   IF (hermaphroditic):
      IF (only one breeding female AND
      ProportionSelfing[p] = 0):
         EXIT BREED();
      END IF
   END IF
   FOR (each female, Dam, in breeding pool):
      Let BreedRand = RAND();
      GETBREEDRATE();
// BreedRate is probability of breeding for the female, given by the user either as a constant, ProportionFemalesBreeding, or as a function of population size and other parameters. See Note 3.
      IF (BreedRate = 0):
         CONTINUE LOOP with next breeding female
      END IF
// Find a mate ...
      IF (hermaphroditic):
         IF (RAND() < ProportionSelfing[p]):
            Let Sire = Dam
         ELSE
            Choose a Sire at random from breeding pool;
            WHILE (Sire is Dam):
               Choose a new Sire;
            END WHILE
         END (selfing) IF/ELSE
      ELSE (not hermaphroditic):
         Choose a Sire at random from the male breeding pool;
```

```
        IF (monogamous):
          WHILE (MaleUsed[Sire]):
            Choose a new Sire;
          END WHILE
          Set MaleUsed[Sire] = TRUE;
// Flag Sire as unavailable for future Dams
        END IF
      END IF/ELSE
// Find the litter size for that pairing ...
      IF (MaximumLitterSize > 0):
        Set CumulativeProbLitterSize[0] = 1 - BreedRate;
        FOR each possible litter size, n:
          Set CumulativeProbLitterSize[n] =
          CumulativeProbLitterSize[n - 1]
          + ProbLitterSize[p][n] * BreedRate;
        END LOOP
        FOR (each litter size, n, in decreasing order):
          IF (BreedRand > CumulativeProbLitterSize
          [n - 1] ):
            Set LitterSize = n;
            BREAK from Litter Size LOOP
          END IF
        END LOOP
      ELSE:
// MaximumLitterSize = 0 is a code for using normal distribution of litter sizes.
        Set LitterSize = MeanLitterSize[p] +
        SDLitterSize[p] * NRAND();
        Set LitterSize = max(0, LitterSize);
        Set LitterSize = min(2 * MeanLitterSize[p],
        LitterSize );
// Truncates symmetrically to avoid creating bias.
        Set IntegerLitter = Largest integer less than
        LitterSize;
        Set Remainder = LitterSize - IntegerLitter;
        IF (RAND() < Remainder):
// Round-off litter size probabilistically.
          Set LitterSize = IntegerLitter + 1;
        ELSE:
          Set LitterSize = IntegerLitter;
        END IF/ELSE
      END IF/ELSE
// Create the offspring ...
      Set Inbreeding = Kinship between Sire and Dam;
      FOR (Offspring from 1 to LitterSize):
        Assign ID, age (= 0), population, alive (= TRUE);
        FOR (each of six loci):
// First locus is neutral, others can have lethals.
          Pick at random an allele from Dam;
          Pick at random an allele from Sire;
        END LOOP
        IF(not hermaphroditic AND RAND() <
        SexRatio):
          Assign sex as male;
        ELSE:
          Assign sex as female;
        END IF/ELSE
// Does offspring live? Offspring mortality is placed here in the code, rather than in the MORTALITY() function, for better speed and lower memory requirements.
        FOR (each non-neutral locus):
          IF (homozygous AND allele is a lethal):
            Offspring dies;
          END IF
        END LOOP
        IF (not yet dead):
          GETDEATHRATE();
          Set SurvivalRate = 1 - DeathRate;
          IF (Inbreeding > 0):
            Set SurvivalRate = exp(-0.50 *
            LethalEquivalents * Inbreeding);
          ENDIF
          IF (RAND() > SurvivalRate):
            Offspring dies;
          END IF
        END IF
        IF (not dead):
          Calculate kinship to every living animal;
// See Ballou (1983) for the method of calculating inbreeding and kinship coefficients.
        END IF
      END (offspring) LOOP
    END (breeding females) LOOP
END FUNCTION BREED()

BEGIN FUNCTION GETBREEDRATE():
    Obtain BreedRate by evaluating fecundity function for
    population and individual parameters;
// Most often, the fecundity function will simply return ProportionFemalesBreeding entered by the user. VORTEX provides the option, however, of making breeding a function of PopulationSize, GeneDiversity, Inbreeding, and other variables. See Note 3.
    ADJUSTRATE(BreedRate, LocalBreedEV[p],
    LocalBreedEVRand, LocalBreedEVNRand);
// Adjust rate for local EV.
    ADJUSTRATE(BreedRate, GlobalBreedEV[p],
    GlobalBreedEVRand, GlobalBreedEVNRand);
// Adjust rate for global EV.
    FOR (each type of catastrophe, c):
      IF (CatastropheFlag[c] = TRUE):
        Multiply BreedRate by
        CatastropheBreedSeverity[p][c];
      END IF
    END LOOP
END FUNCTION GETBREEDRATE()

BEGIN FUNCTION MORTALITY(for population p):
    FOR (each living animal in the population):
      IF (age > 0):
// Infant mortality occurs within the BREED() function, not here.
```

```
      IF (at maximum age):
        Animal dies;
      ELSE:
        GETDEATHRATE();
        IF (RAND() < DeathRate ):
          Animal dies;
        END IF
      END IF/ELSE
    END (age > 0) IF
  END (animal) LOOP
END FUNCTION MORTALITY()

BEGIN FUNCTION GETDEATHRATE():
  Obtain DeathRate by evaluating mortality function for
  population and individual parameters;
// Most often, the mortality function will simply return the
mortality rate entered by the user for the age and sex of the
current animal. VORTEX provides the option, however,
of making mortality a function of PopulationSize,
GeneDiversity, Inbreeding, and other variables.
  ADJUSTRATE(DeathRate, LocalMortEV[p],
  LocalMortEVRand, LocalMortEVNRand);
// Adjust rate for local EV.
  ADJUSTRATE(DeathRate, GlobalMortEV[p],
  GlobalMortEVRand, GlobalMortEVNRand);
// Adjust rate for global EV.
  FOR (each type of catastrophe, c):
    IF (CatastropheFlag[c] = TRUE):
      Let DeathRate = 1 -
      (CatastropheSurvivalSeverity[p][c] *
      (1 - DeathRate));
    END IF
  END LOOP
END FUNCTION GETDEATHRATE()

BEGIN FUNCTION ADJUSTRATE(Rate, EV, EVRand,
EVNRand);
  Determine binomial parameter n for modeling EV;
// See Note 6.
  IF (n < 26):
// Find the adjusted Rate from binomial EV.
    FOR (each BinomialOutcome 0 through n):
      Add BinomialOutcome / n to BinomialProportion;
      Calculate BinomialProbability for
      BinomialProportion;
      Add BinomialProbability to CumulativeBinomial;
      IF ( EVRand < CumulativeBinomial ):
        Set Rate = Binomial Proportion;
        BREAK from LOOP
      END IF
    END LOOP
  ELSE:
// Use Normal distribution for EV, and truncate symmetri-
cally to avoid bias.
    IF (Rate > 0.5):
      Set UpperLimit = 1;
      Set LowerLimit = Rate - (1 - Rate);
    ELSE:
      Set UpperLimit = 2 * Rate;
      Set LowerLimit = 0;
    END IF/ELSE
    Add EV* EVNRand to Rate;
    Let Rate = max(LowerLimit, Rate);
    Let Rate = min(UpperLimit, Rate);
  END IF/ELSE
END FUNCTION ADJUSTRATE();

BEGIN FUNCTION MIGRATE():
  FOR (each living animal):
    IF (not in age range that migrates):
      CONTINUE LOOP with next animal;
    END IF
    IF (not a sex that migrates):
      CONTINUE LOOP with next animal;
    END IF
    Set MigrationRand = RAND();
    Set pSource to population of current animal;
    IF ( MigrationRand > CumulativeMigrationProb
    [pSource][NumberPopulations] ):
      CONTINUE LOOP with next animal;
// Does not migrate
    END IF
    Obtain MigrationDensity by evaluating function, or
    using specified constant parameter;
// See Note 3.
    IF (PopulationSize[pSource] / CarryingCapacity
    [pSource] < MigrationDensity ):
      CONTINUE LOOP with next animal;
    END IF
    Obtain MigrationSurvival by evaluating function, or
    using specified constant parameter;
// See Note 3.
// Find to which population the animal migrates ...
    FOR (up to 10 attempts to enter another
    population):
// The limit of 10 attempts is imposed to prevent an infi-
nite loop from occurring when all populations are at carry-
ing capacity.
      IF (RAND() > MigrationSurvival ):
        Animal dies;
        BREAK from LOOP, CONTINUE with
        next animal;
      END IF
      FOR (each population, pDestination):
        IF ( MigrationRand < CumulativeMigrationProb
        [pSource][pDestination] ):
          BREAK from LOOP;
// Animal will try to migrate to pDestination
      END IF
    END LOOP
    IF (Population pDestination at carrying capacity):
      IF (tried 9 times before to find an open population):
```

```
                Animal dies;
// Never found an open population into which to migrate.
                BREAK from LOOP, CONTINUE with
                next animal;
              END IF
              IF (CumulativeMigrationProb[pDestination]
              [NumberPopulations] = 0):
                Animal dies;
// Cannot migrate away from pDestination.
                BREAK from LOOP, CONTINUE with
                next animal;
              END IF
              Set MigrationRand = RAND();
              Set pSource = pDestination;
// Moves on from population pDestination , old pDestination
becomes current pSource.
              WHILE (MigrationRand >
                CumulativeMigrationProb[pSource]
                [NumberPopulations]):
                Set MigrationRand = RAND();
// Must migrate somewhere, so draw a new random
number.
            END WHILE
          END IF
        END LOOP
        Change animal's population to pDestination;
        Adjust tallies of population sizes;
// Increment size of pDestination , decrement size of pSource.
    END animal LOOP
END FUNCTION MIGRATE()

BEGIN FUNCTION HARVEST(for population p):
    FOR (each age, x):
// HARVEST() lumps all animal above breeding age as a
single class.
        IF (NumberMales[p][x] <=
        NumberMalesToBeHarvested[p][x]):
            All males age x die;
        ELSE
            WHILE (number harvested <
            NumberMalesToBeHarvested[p][x]):
                Choose at random a living male in age class x;
                Male dies;
            END WHILE
        END IF/ELSE
    END LOOP
    FOR (each age, x):
        IF (NumberFemales[p][x] <=
        NumberFemalesToBeHarvested[p][x]):
            All females age x die;
        ELSE
            WHILE (number harvested <
            NumberFemalesToBeHarvested[p][x] from age class):
                Choose at random a living female in age class x;
                Female dies;
            END WHILE
        END IF/ELSE
    END LOOP
    Adjust tallies of population size;
END FUNCTION HARVEST()

BEGIN FUNCTION SUPPLEMENT(for population p):
    FOR (each age, x, up to MaleBreedingAge):
        WHILE (number males created <
        NumberMalesToBeSupplemented[p][x]):
            Create a male, assigning ID, age, sex, alleles,
            population;
            Set kinships to all other animals = 0;
            Set Inbreeding = 0;
        END WHILE
    END LOOP
    FOR (each age, x, up to FemaleBreedingAge):
        WHILE (number females created <
        NumberFemalesToBeSupplemented[p][x]):
            Create a female, assigning ID, age, sex, alleles,
            population;
            Set kinships to all other animals = 0;
            Set Inbreeding = 0;
        END WHILE
    END LOOP
END FUNCTION SUPPLEMENT()

BEGIN FUNCTION CALC_GENETIC_METRICS(for
population p):
    FOR (each living animal in the population):
        Increment NumberAlleleCopies[a] for each of the
        two alleles, a, at a neutral locus;
        IF (allele 1 is same as allele 2):
            Increment NumberHomozygotes;
        END IF
    END LOOP
    FOR (each allele, a, of the neutral locus):
        IF ( NumberAlleleCopies[a] > 0 ):
            Increment NumberExtantAlleles;
            Add 0.25 * (NumberAlleleCopies[a] /
            PopulationSize[p]) * (NumberAlleleCopies[a]
            / PopulationSize[p]) to ExpectedHomozygosity[p];
        END IF
    END LOOP
    Set GeneDiversity[p] = 1 – ExpectedHomozygosity[p];
    Set ObservedHeterozygosity[p] = 1 - (NumberHomozygotes /
    PopulationSize[p]);
    FOR (each living animal in the population):
        FOR (each non-neutral locus):
            IF (allele 1 at the locus is a lethal):
                Increment NumberLethals;
            END IF
            IF (allele 2 at the locus is a lethal):
                Increment NumberLethals;
            END IF
        END locus LOOP
    END animal LOOP
```

Set *LethalFrequency[p]* = *NumberLethals* / *PopulationSize[p]*;
END FUNCTION CALC_GENETIC_METRICS()

Note 1: Random integers from 0 to 64K are generated by the algorithm given by Kirkpatrick and Stoll (1981). The C code was modified from Maier (1991). Random real numbers between 0 and 1 are produced by first generating a random integer between 0 and 64K, and then dividing that integer by 64K. Random numbers from a normal distribution, with mean = 0 and SD = 1 are generated by the polar algorithm supplied by Latour (1986). Binomially distributed numbers are generated by first calculating the cumulative probability distribution for the discrete outcomes of the desired distribution, then generating a random real number, and then assessing which binomial outcome covers the portion of the distribution encompassing the random real number.

Note 2: VORTEX asks for the effects of inbreeding to be entered as a number of "lethal equivalents" per diploid animal, with further specification of what proportion of this genetic load is due to recessive lethal alleles vs other genetic effects (such as overdominance). Recessive lethal alleles are modeled such that the death of animals homozygous for lethal alleles will reduce the frequency of the lethals and thereby reduce the average effects of future inbreeding. The proportion of inbreeding depression not due to lethal alleles is modeled as an impact on fitness that follows a negative exponential equation (Morton et al. 1956), and is not reduced during generations of inbreeding.

Note 3: For rates which can be specified as functions of age, sex, inbreeding, population size, gene diversity, year, and population, the rate to be used is determined by evaluating the function specified by the user. If the user enters a fixed constant for the rate, as is usually the case, then the function simply returns that constant. However, the user can specify a mathematical formula that defines a demographic rate as being density-dependent, or a function of other population parameters. For example, fecundity could be specified to decline in older age classes, adult mortality could be specified to increase with inbreeding, or habitat (carrying capacity) could be specified to decrease over time. The algorithms for parsing and evaluating user-defined rate functions (e.g., the first step of functions GETBREEDRATE() and GETDEATHRATE()) are not given in the pseudo-code.

Note 4: The proportion of males in the breeding pool can be entered directly, or indirectly in the form of the proportion of males that breed or as the average number of litters per breeding male. If the proportion in the breeding pool is given indirectly, VORTEX will assume that the distribution of male reproductive success follows a Poisson distribution. The proportion of males in the breeding pool is then calculated by solving the following equations for the unknowns:

LittersPerMale = *ProportionFemalesProducingLitters* * (*NumberAdultFemales* / *NumberAdultMales*) [Note: Adult sex ratio is determined from stable age distribution.]

LittersPerMale = *ProportionMalesInBreedingPool* * *LittersPerMaleInBreedingPool*

ProportionMalesBreeding = *ProportionMalesInBreedingPool* * (1 - exp(-*LittersPerMaleInBreedingPool*))

This last equation adjusts for the fact that in any given year some males in the breeding pool will not happen to be successful breeders (the zero class of the Poisson distribution).

Note 5: The occurrence of probabilistic events is determined by a random number generator. The event is deemed to occur if a random number between 0 and 1 is less than the probability of occurrence for that event.

Note 6: Environmental variation (EV) in breeding and in each mortality rate is modeled as a binomial distribution or as a normal distribution, depending on whether the magnitude of EV is large. The user specifies a mean and standard deviation for each rate. The binomial distribution that has a standard deviation closest to the desired EV is determined by solving the equation for the binomial variance, $V = p * (1 - p) / n$, for the parameter n when given the mean, p, and variance, $V = SD^2$. The parameter n is then rounded to the nearest whole number. If $n < 26$, the binomial distribution with parameters p and n is used for EV. Because of the rounding step necessary to produce the discrete binomial distribution, this distribution will often have a slightly different variance than that entered by the user. If $n > 25$, the normal distribution with mean p and variance V will be used to model EV. In such cases, the normal distribution very closely approximates the binomial distribution.

The binomial distribution is restricted to the interval 0 to 1, and it fits well the distribution of demographic rates across years observed in some natural populations (e.g., Lacy 1993). The PVA program INMAT (Mills and Smouse 1994) uses the related beta distribution for this purpose, and it too is restricted to the biologically meaningful 0 to 1 interval. In contrast, the normal distribution extends infinitely in both directions, although the tails beyond 0 and 1 are typically very small in those cases for which VORTEX uses a normal distribution to model EV. For example, if $p = 0.5$ and $SD = 0.1$ (so that the binomial parameter $n = 25$, the limiting case for VORTEX to use the normal approximation), then the area of the normal distribution outside of the 0-1 range is < 0.000001. When modelling EV as a normal distribution, the distribution must be truncated at 0 and 1. To avoid creating any bias in the mean demographic rate as a result of this truncation, VORTEX always truncates the distribution symmetrically.

For example, if the mean is $p = 0.3$, VORTEX truncates the distribution at 0.0 and 0.6. This truncation will cause the *SD* of the distribution to be very slightly less than that entered by the user.

Some PVA models use continuous distributions such as the normal or log normal to represent EV even when EV is large. In such cases, the necessary truncations can cause EV to be substantially less than intended by the user. Moreover, if the truncation is not symmetric, then the mean demographic rate generated by the model can be strongly biased away from the input parameter.

References

Ballou, J. D. 1983. Calculating inbreeding coefficients from pedigrees. – In: Schonewald-Cox, C. M. et al. (eds), Genetics and conservation: a reference for managing wild animal and plant populations. Benjamin/Cummings, Menlo Park, California, pp. 509–520.

Burgman, M. A., Ferson, S. and Akçakaya, H. R. 1993. Risk assessment in conservation biology. – Chapman and Hall.

Caswell, H. 1989. Matrix population models: construction, analysis and interpretation. – Sinauer.

Caughley, G. and Gunn, A. 1996. Conservation biology in theory and practice. – Blackwell.

Kirkpatrick, S. and Stoll, E. 1981. A very fast shift-register sequence random number generator. – J. Comp. Phys. 40: 517.

Lacy, R. C. 1993. VORTEX: A computer simulation model for population viability analysis. – Wildl. Res. 20: 45–65.

Latour, A. 1986. Polar normal distribution. – Byte, August 1986: 131–132.

Lindenmayer, D. B., Lacy, R. C. and Pope, M. L. 2000. Testing a simulation model for population viability analysis. – Ecol. Appl. 10: 580–597.

Maier, W. L. 1991. A fast pseudo random number generator. – Dr. Dobb's Journal, May 1991: 152–157.

Miller, P. S. and Lacy, R. C. 1999. VORTEX Ver. 8 users manual. A stochastic simulation of the simulation process. – IUCN/SSC Conservation Breeding Specialist Group, Apple Valley, Minnesota.

Mills, L. S. and Smouse, P. E. 1994. Demographic consequences of inbreeding in remnant populations. – Am. Nat. 144: 412–431.

Morton, N. E., Crow, J. F. and Muller, H. J. 1956. An estimate of the mutational damage in man from data on consanguineous marriages. – Proc. Nat. Acad. Sci. USA 42: 855–863.

Pielou, E. C. 1977. Mathematical ecology. - Wiley.

Starfield, A. M. and Bleloch, A. L. 1986. Building models for conservation and wildlife management. – MacMillan.