

Lab 12: Estimating Confidence Limits for Population Growth Rates in Matlab

Objectives: The goal of a demographic analysis is often to estimate the rate of population change or other matrix properties. Because the vital rates are often based on imperfect data, the estimates of λ are estimated with uncertainty. As with most statistics, it is desirable to quantify the uncertainty and standard errors and confidence intervals are the traditional tools for doing so. Quantifying the uncertainty in λ permits statistical inference for important ecological questions: Is the rate of population change significantly different from 1? Is the rate of population change for two populations significantly different? The goal of this exercise is to present two methods for calculating the variance of estimates of population growth rate $Var(\hat{\lambda})$: an analytical method based on a series approximation and a randomization or bootstrapping procedure.

I. Series approximation. One approach to estimating the standard error of a demographic statistic relies on a series expansion, sometimes called the “delta method”. The first order approximation to the variance of λ is given by the following formula:

$$\text{eqn. 1} \quad Var(\hat{\lambda}) \approx \sum \sum Cov(a_{ij}, a_{kl}) \times (\partial\lambda / \partial a_{ij}) \times (\partial\lambda / \partial a_{kl})$$

where $Var(\hat{\lambda})$ is the variance in the rate of population change (the little hat ^ indicates that $\hat{\lambda}$ is an estimate), \approx indicates that this formula is an approximation only, $Cov(a_{ij}, a_{kl})$ is the variance-covariance matrix for the matrix elements and $\partial\lambda / \partial a_{ij}$ and $\partial\lambda / \partial a_{kl}$ are the sensitivity matrix for the mean matrix expressed as row and column vectors. This is the same formula that is used in calculating the contributions to population growth rate for a random design Life-table Response Experiment (LTRE). The standard error and 95% CI of $\hat{\lambda}$ can be estimated from:

$$\text{eqn. 2} \quad SE(\hat{\lambda}) = \sqrt{Var(\hat{\lambda})}$$

$$\text{eqn. 3} \quad 95\%CI(\hat{\lambda}) = \hat{\lambda} \pm 1.96 \times SE(\hat{\lambda})$$

The series approximation to $Var(\hat{\lambda})$ assumes that the variances are small and its use in constructing confidence intervals assumes that $\hat{\lambda}$ is normally distributed (i.e., symmetrical).

a) Using Series Approximation to Calculate the Confidence Limits of λ for Killer Whales:

We can use these calculations in Matlab for the Killer Whale example. Startup Matlab. Click on File | New | m-file to create a new m-file. Enter the projection matrices for the three southern pods of the killer whale from the Appendix of Brault and Caswell's paper. Be sure that you use zeros and not capital “O” throughout:

```
J01=[0 0.0067 0.1632 0; 0.9535 0.8827 0 0; 0 0.0802 0.9586 0; 0 0.0414 0.9752];
K01=[0 0.0062 0.1737 0; 1.0000 0.9020 0 0; 0 0.0694 0.9582 0; 0 0.0418 0.9855];
L01=[0 0.0037 0.0988 0; 0.9562 0.9030 0 0; 0 0.0722 0.9530 0; 0 0.0406 0.9798];
```

Click on File | Save to save the file in the work directory. Name it “south.m”. Close this window and go back to Matlab. Make sure that the Current Directory on the line above the

Command Window reads: “c:\matlabR12\work”. Go back to File | Open and retrieve the south.m file by typing ‘south’.

The next lines are some of the same commands that you used last week to calculate the matrix properties. You will need to enter these again and you can go through them to remind yourself of the calculations

Step 1. Calculate the average of the three matrices.

```
mat = (J01 + K01 + L01) / 3
```

Step 2. These Matlab commands produce matrices of the eigenvalues (dmat) and the eigenvectors (wmat and vmat) for the average matrix.

```
[wmat, dmat] = eig(mat)
vmat = conj(inv(wmat))
```

Step 3. The next commands calculate the dominant eigenvalue or λ (lambda1).

```
lambdiag = diag(dmat);
lambvec = sort(abs(lambdiag));
dim = size(lambvec,1);
lambda1 = lambvec(dim)
```

Step 4. The next commands calculate the stable age distribution and reproductive values from the eigenvector matrices.

```
imax = find(lambdiag==max(lambdiag));
rawage = wmat(:,imax);
sad = rawage./sum(rawage)
rawrv = vmat(imax,:)'
rv = rawrv / rawrv(1,1)
```

Step 5. And the last step is to calculate the sensitivity matrix for the average matrix.

```
sens = (rv*sad') ./ (sum(rv.*sad))
```

Now we are ready to do the calculations for each of the pieces of eqn. 1 above.

Step 6. Calculate the variance covariance matrix.

```
matcol = [J01(:) K01(:) L01(:)]
covmat = cov(matcol')
senscol = sens(:)
```

What are these commands doing? The first line creates a matrix where each of the pod-specific matrices have been converted to a vector column. The second line calculates the variance-

covariance matrix for the starting matrix. If the original matrix is $n \times n$ then the variance-covariance matrix will be $n^2 \times n^2$. Recall that the diagonal elements are the variances of the vital rates whereas the off-diagonal elements are the covariances for all the possible pairs of vital rates. The last line converts the sensitivity matrix to a column vector.

Step 7. Now we are ready to calculate the variance of the rate of population change using eqn. 1.

```
varlamb = covmat.*(senscol*senscol')
varlamb = sum(sum(varlamb))
```

Line 1. The second part of this command multiplies the sensitivity column vector against itself as a row vector (' transposes the vector). The first part of the command multiplies the variance covariance matrix against this new matrix on an element by element basis.

Line 2. This command reduces the matrix to a single scalar sum.

Step 8. Calculate the standard error and 95%CI of the rate of population change using eqns. 2-3.

```
SElamb = sqrt(varlamb)
lambda1
low95 = lambda1 - 1.96*SElamb
high95 = lambda1 + 1.96*SElamb
```

These commands calculate the standard error of $\hat{\lambda}$, print the rate of population growth rate on the screen and then calculate the 95% confidence interval. Save the south.m file and close it.

II. Boot-strapping. Bootstrap sampling (or Monte Carlo sampling) are a randomization technique that permits estimation of parameters with unknown distributions. It is a simple method but requires a computer because it is computationally intensive. One advantage of this approach is that it is not constrained by the assumptions of the series approximation method (small variances, normal distribution of λ). Another feature is that bootstrapping can be used to calculate standard errors and confidence intervals for any demographic statistic, and not just λ . Bootstrap methods are also the basic approach for creating stochastic simulation models, which are an extension of the deterministic models that we have been studying.

Bootstrap resampling requires an ability to take random draws from statistical distributions, including the normal, lognormal, beta, and uniform statistical distributions. Recall that fecundity is unconstrained and can vary from 0 to extremely large numbers. If a mean and a standard deviation are available, the normal distribution can be used to model these rates. On the other hand, survival and other transition rates range from 0-1. If a mean and a standard deviation are available, a handy distribution for modeling these rates is the beta distribution because it is bounded between these two extremes. Finally, if only a range of values is available, a uniform or flat distribution can be used. The macros on the following pages provide examples of how to obtain draws from several common statistical distributions. These draws could then be combined to calculate lower-level demographic rates or matrix elements.

a) Resampling from a normal distribution:

Create an m-file with the following lines:

```
% Draws from a normal distribution;
% mean = 0.9321 and std = 0.1828;
P=10;

for n=1:P;
    draw(n) = 0.9321 + 0.1828*randn;
end;

out = [draw(1:P)]';
mean = mean(out)
std = std(out)

clear P draw n mean std out
```

Lines 1-2. The first two lines are documentation for the file, the % symbol tells Matlab to read the line as a comment.

Line 3. P is the number of trials that you want to run, start with ten to illustrate the process.

Lines 4 and 6. The for | end is a loop that will repeat the analysis for the number of trials (i.e., P).

Line 5. Is the heart of the routine. To obtain draws from a normal distribution, a random number function generator is used. randn gives random draws from a normal distribution with a mean of zero and a standard deviation of one. To scale these to our own distribution, the standard deviation is multiplied by randn and added to the mean.

Lines 6-8. This line collects all of the draws into a matrix named out and then calculates the mean and standard deviation of these draws as a check. Sampling theory predicts that for P=10 it is likely the values will differ from the input rates, for P=1000 or P=10000 they should converge to the input rates.

Line 9. This line is a bookkeeping statement and is critical. For each successive run, you need to clear the variables and matrices before running the macro again.

b) Resampling from a beta distribution:

The normal distribution could be used to generate random draws for a survival rate. For example, if your mean survival rate was 0.8 and the standard deviation was 0.2 you could use the macro above to get reasonable results. The problem in using the normal distribution is that some of the draws might be greater than one. In these cases, you would need to use some decision rule about whether you would truncate these to 1 or draw again. A better solution is to use a beta distribution where all the draws will be bounded 0-1. Use of the beta distribution requires a little trick. For the normal distribution, we simply used the mean and the standard deviation. Draws from the beta distribution require the distribution to be described by two 'shape' parameters. A necessary first step then is to calculate these shape parameters from the mean and the standard deviation. Type the following lines into an m-file.

```

% Draws from a beta distribution;
% mean = 0.9234 and std = 0.0716;

P=10;

syms a b;
c = 0.9234;
d = 0.0716^2;
[a,b] = solve((a/(a+b))-c, ((a*b)/((a+b)^2*(a+b+1)))-d);
a=double(a)
b=double(b)

for n = 1:P;
    beta(n) = betarnd(a,b);
end

out= [beta(1:P)]'
mean = mean (out)
STD = std (out)

clear P a b c d beta n mean std out

```

Lines 1-2. The first two lines are documentation for the file.

Line 3. P is the number of trials that you want to run, start with ten to illustrate the process.

Lines 4- 6. These lines create the two shape parameters a and b and then set the values for the mean c and the variance d. Recall that the variance is equal to the standard deviation squared.

Line 7. This is a bit ugly but is the step that calculates the shape parameters for the beta distribution. This trick is based on substituting the values of c and d into two equations and then solving for the values of a and b that satisfy both equations.

Lines 8-9. Because a and b were defined as symbols, Matlab handles the values in a fractional notation. The double command converts these values to a decimal notation.

Lines 10 and 12. The for |end is a loop that will repeat the analysis for the number of trials (i.e., P).

Line 11. Is the heart of the routine. To obtain draws from a beta distribution, betarnd takes random draws from a beta distribution described by the two shape parameters a and b.

Lines 12-14. This line collects all of the draws into a matrix named out and then calculates the mean and standard deviation of these draws as a check. Inspect the values in out. Note that even though the mean was close to the boundary of one, all of the draws are ≤ 1 .

Line 15. This line is a bookkeeping statement and is critical. For each successive run, you need to clear the variables and matrices before running the macro again.

c) Resampling from a uniform distribution

```

% Draws from a uniform distribution;
% range = 0.3 to 0.8;
P=10;

```

```

for n=1:P;
    draw(n) = 0.3 + (0.8-0.3)*rand;
end;

out = [draw(1:P)]';
min = min(out)
max = max(out)

clear P draw n mean std out

```

Lines 1-2. The first two lines are documentation for the file, the % symbol tells Matlab to read the line as a comment.

Line 3. P is the number of trials that you want to run, start with ten to illustrate the process.

Lines 4 and 6. The for | end is a loop that will repeat the analysis for the number of trials (i.e., P).

Line 5. Is the heart of the routine. To obtain draws from a uniform or flat distribution, a random number function generator is used. rand gives random draws from a uniform distribution bounded between 0 and 1. To scale these to the range of our values, the difference between the maximum and minimum values are multiplied by rand and added to the minimum.

Lines 6-8. This line collects all of the draws into a matrix named out and then calculates the min and max of these draws as a check.

c) Using Bootstrapping to Calculate the Confidence Limits of λ for Lesser Kestrels:

The following program illustrates the use of bootstrap resampling within Matlab. It uses the resampling methods for the normal and beta distributions above to generate random estimates for the five vital rates that comprise the kestrel matrix. It then calculates λ for a matrix comprised of each random set of vital rates. The 95% confidence intervals can then be directly observed from the distribution of λ -values generated from the random draws. The statistical commands in Matlab are fairly limited so this program also shows how the output can be directed into an ASCII file that can be then analysed or graphed with your favorite statistical software program. Enter the following lines of code and play with the program to learn the mechanics of what it is doing. Ask questions if any of this code is unclear!

```

% Bootstrap confidence limits of lambda for the kestrel matrix;

P = 10;

% Start of the loop;
for n = 1:P;

% Counter for the screen;
PercentDone=n/P

```

```

% bd, Number of female offspring produced using normal
distribution (mean = 0.9321 SD = 0.1828);
bd(n) = (0.9321 + 0.1828 * randn);

% co, Yearling probability of breeding using beta distribution
(mean = 0.3847, SD = 0.0716);
syms a b;
c = 0.3847;
d = 0.0716^2;
[a,b] = solve((a/(a+b))-c, ((a*b)/((a+b)^2*(a+b+1)))-d);
a=double(a);
b=double(b);
co(n) = betarnd (a,b);
clear a b c d;

% ca, Adult probability of breeding using beta distribution
(mean = 0.9250, SD = 0.0811);
syms a b;
c = 0.9250;
d = 0.0811^2;
[a,b] = solve((a/(a+b))-c, ((a*b)/((a+b)^2*(a+b+1)))-d);
a=double(a);
b=double(b);
ca(n) = betarnd (a,b);
clear a b c d;

% so, Fledging to adult survival probability using beta
distribution (mean = 0.3409, SD = 0.1200);
syms a b;
c = 0.3409;
d = 0.1200^2;
[a,b] = solve((a/(a+b))-c, ((a*b)/((a+b)^2*(a+b+1)))-d);
a=double(a);
b=double(b);
so(n) = betarnd (a,b);
clear a b c d;

% sa, Adult survival probability using beta distribution (mean =
0.7101, SD = 0.0726);
syms a b;
c = 0.7101;
d = 0.0726^2;
[a,b] = solve((a/(a+b))-c, ((a*b)/((a+b)^2*(a+b+1)))-d);
a=double(a);
b=double(b);
sa(n) = betarnd (a,b);
clear a b c d;

```

```

% Parameterize the matrix and calculate lambda;
mat = [co(n)*bd(n)*so(n) ca(n)*bd(n)*so(n); sa(n) sa(n)];
[wmat,dmat] = eig(mat);
lam = diag(dmat);
imax=find(lam==max(lam));
lambda(n) = lam(imax);

end

% Collect lambda from each set of random draws into a matrix
named out;
out = [lambda(1:P)]';
mean=mean(out)
median=median(out)

% Pull the 90% or 95% CI directly from the distribution of
lambda values;
out=sort(out);
Low10 = out(P*0.1)
High90 = out(P*0.9)

% The output can also be directed into an external ASCII file;
% For some reason, the matrix needs to be flipped back to a row
vector first;
out=out';
% Create the external file, 'output.txt' is the filename, 'w'
creates or overwrites previous files;
fid = fopen('output.txt','w');
% Printing to the output file, %f is a format for fixed decimal
places where 2.6 is
% a minimum of 2 digits and 6 places after the decimal, \n puts
values in a column;
fprintf(fid,'%2.6f\n', out);
fclose(fid);

% Bookkeeping between runs;
clear;

```