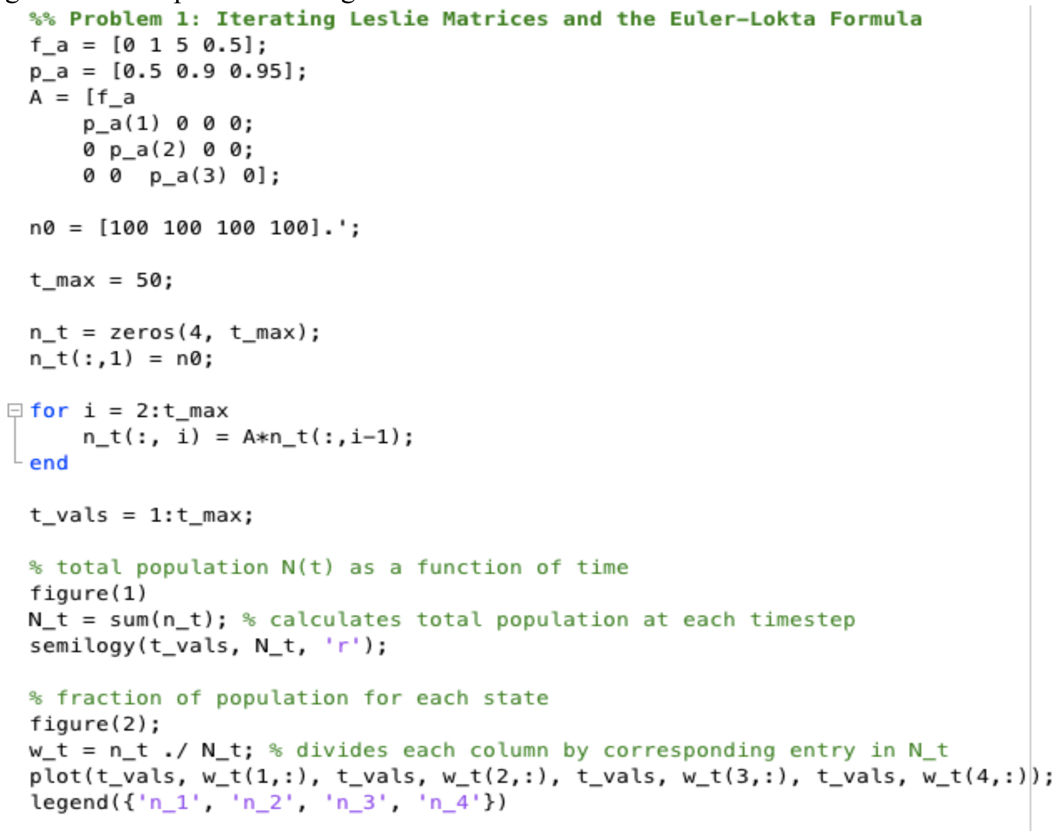
Zachary McNulty
AMATH 422

Problem Set #1

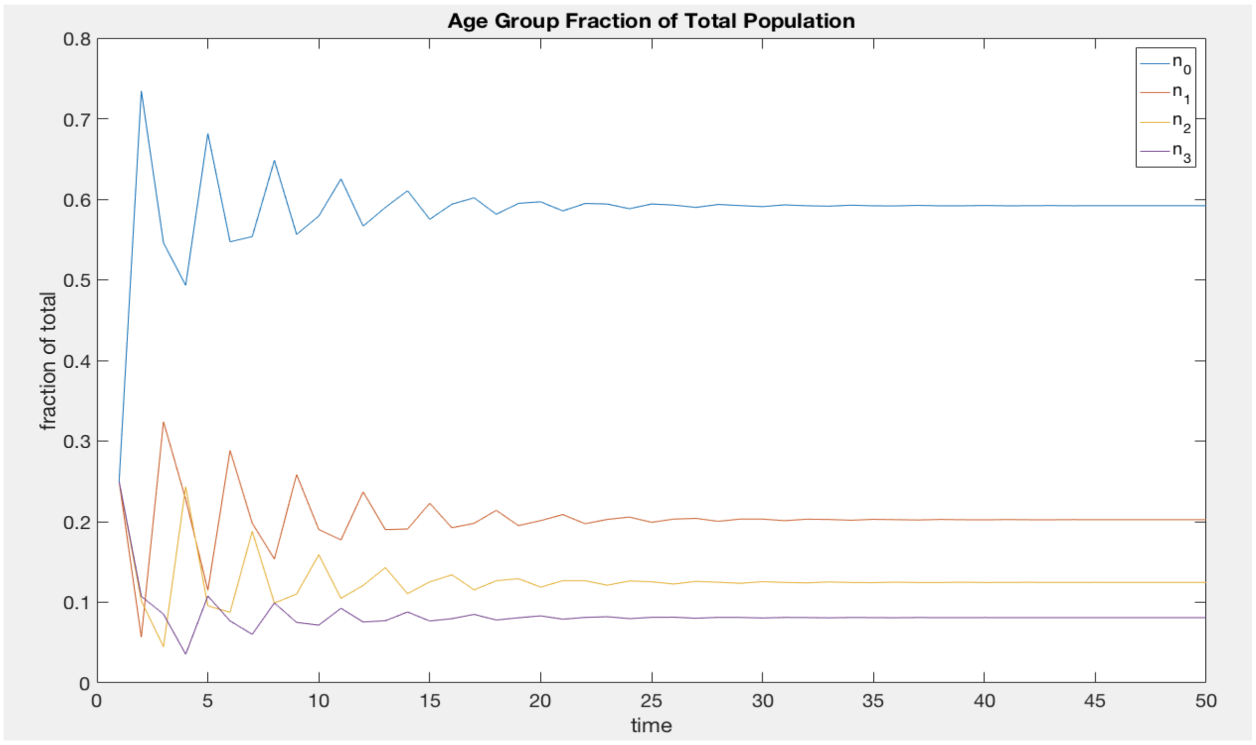
Problem 1: Iterating Leslie Matrices and the Euler-Lokta Formula

In this problem, we were attempting to find the growth rate of our population in two separate ways: using the Euler-Lokta formula and using polynomial fitting. We first iterate over several timesteps in order to get some data points for our growth model.

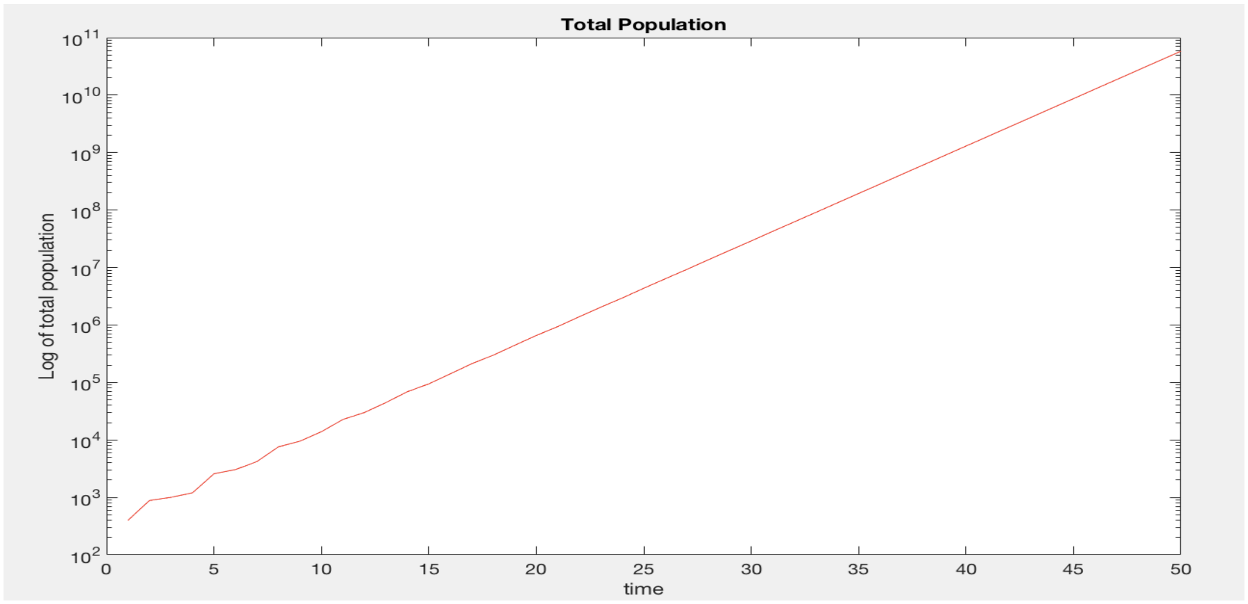


```
%% Problem 1: Iterating Leslie Matrices and the Euler-Lokta Formula
f_a = [0 1 5 0.5];
p_a = [0.5 0.9 0.95];
A = [f_a
    p_a(1) 0 0 0;
    0 p_a(2) 0 0;
    0 0  p_a(3) 0];

n0 = [100 100 100 100].';

t_max = 50;

n_t = zeros(4, t_max);
n_t(:,1) = n0;

for i = 2:t_max
    n_t(:, i) = A*n_t(:,i-1);
end

t_vals = 1:t_max;

% total population N(t) as a function of time
figure(1)
N_t = sum(n_t); % calculates total population at each timestep
semilogy(t_vals, N_t, 'r');

% fraction of population for each state
figure(2);
w_t = n_t ./ N_t; % divides each column by corresponding entry in N_t
plot(t_vals, w_t(1,:), t_vals, w_t(2,:), t_vals, w_t(3,:), t_vals, w_t(4,:));
legend({'n_1', 'n_2', 'n_3', 'n_4'})
```

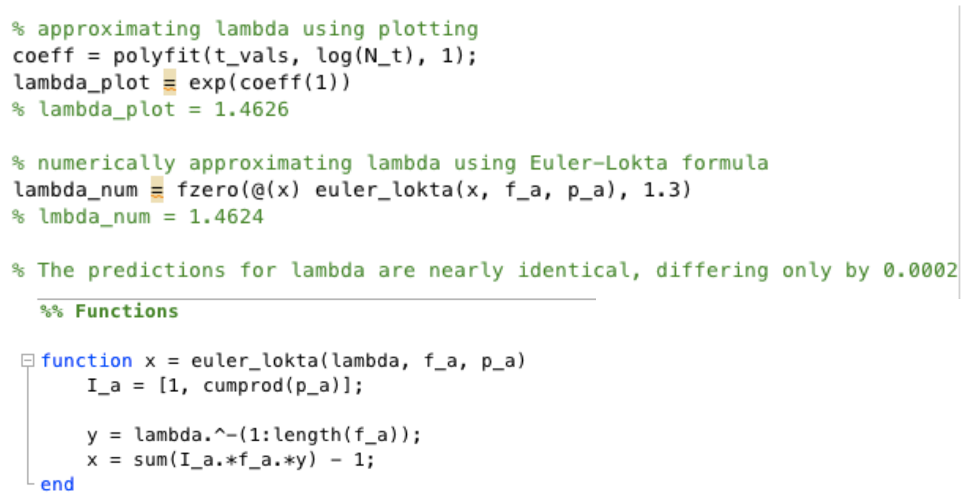
In order to get some idea of the age group dynamics our model generates, we plot the fraction of each age group in the population. We can see that, although there is some initial oscillation, the system quickly reaches a steady-state where the earliest age-group (n_0) dominates the overall population.



Looking at a log plot of the overall population, we see a very linear pattern, suggesting our growth is exponential. As such, we can approximate the growth rate, lambda, of this population using the slope of this line.



Here, we calculate the slope of the line using the polyfit() command and convert it into our growth rate lambda so that our population matches the form N(t) = lambda^t. This gives lambda = 1.4626. To verify this solution, we numerical calculate the zero of the Euler-Lokta equation related to this problem. We get a very similar value for lambda, 1.4624. As lambda > 1, our population size is increasing, as we see in our graphs as well.



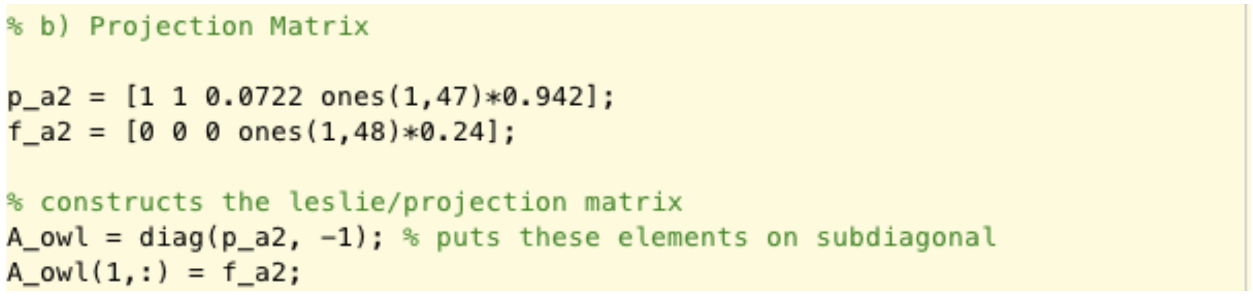
```
% approximating lambda using plotting
coeff = polyfit(t_vals, log(N_t), 1);
lambda_plot = exp(coeff(1))
% lambda_plot = 1.4626

% numerically approximating lambda using Euler-Lokta formula
lambda_num = fzero(@(x) euler_lokta(x, f_a, p_a), 1.3)
% lmbda_num = 1.4624

% The predictions for lambda are nearly identical, differing only by 0.0002
```

```
%% Functions

function x = euler_lokta(lambda, f_a, p_a)
    I_a = [1, cumprod(p_a)];

    y = lambda.^-(1:length(f_a));
    x = sum(I_a.*f_a.*y) - 1;
end
```

Problem 2: Owls!

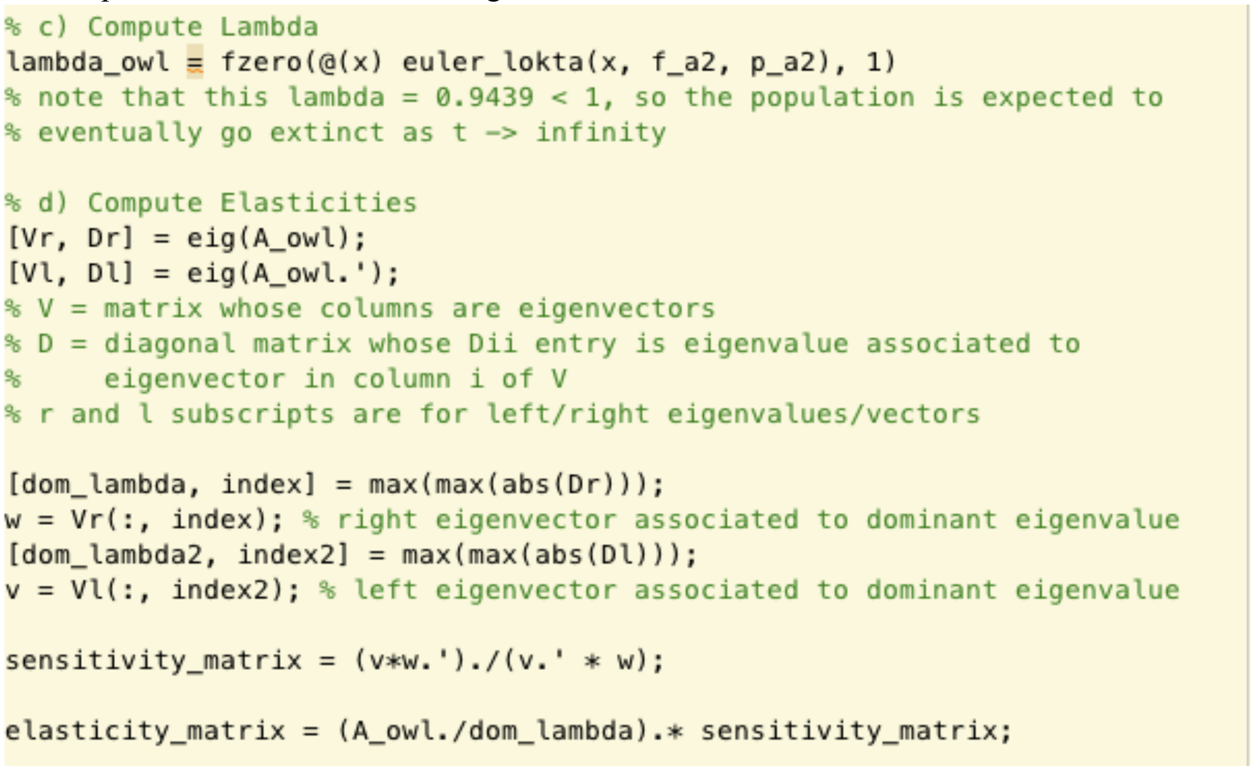
a)

Because the fecundity of all owls below the age of 3 is zero. Looking at the Euler-Lokta formula, we see that this will turn the first few terms of the summation to zero, and therefore the populations these age groups will not affect the calculation of lambda. Thus, simply knowing I_3 and the adult survival rate is sufficient. Here, I simply create the required projection matrix, placing the p_a values on the sub-diagonal and the f_a values in the first row.



```
% b) Projection Matrix

p_a2 = [1 1 0.0722 ones(1,47)*0.942];
f_a2 = [0 0 0 ones(1,48)*0.24];

% constructs the leslie/projection matrix
A_owl = diag(p_a2, -1); % puts these elements on subdiagonal
A_owl(1,:) = f_a2;
```

I compute lambda here in the same way I computed it in problem 1: finding the zero of the Euler-Lokta formula. After doing so, I find the dominant eigenvalue and calculate the elasticities of each parameter in relation to this eigenvalue.



```
% c) Compute Lambda
lambda_owl = fzero(@(x) euler_lokta(x, f_a2, p_a2), 1)
% note that this lambda = 0.9439 < 1, so the population is expected to
% eventually go extinct as t -> infinity

% d) Compute Elasticities
[Vr, Dr] = eig(A_owl);
[Vl, Dl] = eig(A_owl.');
% V = matrix whose columns are eigenvectors
% D = diagonal matrix whose Dii entry is eigenvalue associated to
%     eigenvector in column i of V
% r and l subscripts are for left/right eigenvalues/vectors

[dom_lambda, index] = max(max(abs(Dr)));
w = Vr(:, index); % right eigenvector associated to dominant eigenvalue
[dom_lambda2, index2] = max(max(abs(Dl)));
v = Vl(:, index2); % left eigenvector associated to dominant eigenvalue

sensitivity_matrix = (v*w.')./(v.' * w);

elasticity_matrix = (A_owl./dom_lambda).* sensitivity_matrix;
```

I notice that the elasticities for all the values of f_a are fairly similar (although slightly decreasing as a increases) except for the first 3 elasticities (associated to f_0, f_1, f_2) which are all zero. It makes sense that these first three are zero because $f_0 = f_1 = f_2 = 0$, so a proportional increase will not do anything. It also makes sense that the other values of f_a are similar because f_a is constant from ages 3 to 50, and as the survival rate is so high the age group sizes do not drastically differ (although will differ significantly, leading to the observed decrease in elasticity of f_a as higher ages). The elasticities for the p_a values, however, consistently decrease as a increases, and are much larger than any of the values of f_a. This makes sense because an increase in survivorship at early ages effects the age-group sizes of many other age groups, and as all ages have the same f_a this increases overall fecundity, while an increase in survivorship at older ages does not affect as many age groups. With this knowledge, the best course of action would be to attempt to increase the survival fraction of young owls, Since l_3 is already so low, a proportional increase in survival for owls in this range will be much easier (and more effective) compared to trying to improve the 95% survival rate of owls above the age of 3.

Problem 4: MATLAB Programming Tips

1)lookfor string:
using this command in the command window, it will search for and output the names of all functions where the given string appears in the doc string of the function name. Adding the -all parameter expands this search even further (lookfor string -all).

2)
If you are running some commands in the command window, you can use the up arrow key to give a list of previous commands there so you won't have to retype them again or you can see what you previously called.

3) Logical arrays

Logical arrays are special vectors/matrices that contain only 1's (true) and 0's (false). They have their own type, so you cannot just generate a logical array doing x = [1 0 0 0 1 1 1], you have to cast the type:
x = logical (x = [1 0 0 0 1 1 1]). Mostly, these arrays are generated when you do element-wise logical comparisons between two vectors. The comparison (== <= >= ~=) must be between two vectors of the same length, in which case the comparison is element by element, or between a vector and a scalar. For example

[1 2 3] == [1 1 1] --> returns the logical array [1 0 0] as when comparing element by element, only first pair is equal.
[1 2 3] >= 2 --> returns the logical array [0 1 1]

Now, why do you care? Well, logical arrays are super helpful for extracting specific elements of vectors/arrays. When you use a logical array as an index (i.e. A(logical array)), it returns only the elements that are true within the logical array. Using this, we can extract elements that meet specific constraints. For example,
v(v >= 4) returns a vector of all the elements that are >= 4 in v. If you were interested in the indices rather than the elements itself, use find(), which returns a list of all the non-zero (true) indices of a logical array ---> find([1 9 7 4] == 7) returns 3 and
v(find(v >= 4)) is equivalent to v(v >= 4)

Problem 5: Project Warm-Up

a) citation

Prado, Kerr, 2008. Evolution of restraint in bacterial biofilm under
nontransitive competition. Evolution 62-3,538-548

b) model purpose

To explore how factors such as the cost of resistance and level of
toxicity evolve in spatially structured habitats. Furthermore, the
paper aims to investigate the effect non-transient, circular competition has
in this localized setting. Explore the process of non-transient
competition in bacteria.

c) state variables

There is an L x L grid of nodes/bacteria colonies, where each node is
one of the following
  S = sensitive; not resistant to the toxin of producer
  P = producer; makes toxin
  R = resistant; resistant to toxin of producer
  E = empty; no bacteria within this node
Once filled, a bacteria colony has a specific probaility of dying at each
stage and leaving its node empty (E), represented by delta i where i is in {S, P, R}
Each empty node has a chance of being inhabited by a given strain that is
proportional to the number of neighbors the empty square has of that
strain

Cells are given the opportunity to evolve using a variable g which can
randomly fluctuate. What g relates to depends on the cell type. For an R
cell, g relates to the death rate. Random mutations, with a given probability, can increase or
decrease this rate in offspring

d) one simplifying assumption
   * the death rate of sensitive cells is a linear function with respect
   to the number of nearby toxic cells