CS1132 Fall 2011 Assignment 1

Adhere to the Code of Academic Integrity. You may discuss background issues and general strategies with others and seek help from course staff, but the implementations that you submit must be your own. In particular, you may discuss general ideas with others but you may not work out the detailed solutions with others. It is never OK for you to see or hear another student's code and it is never OK to copy code from published/Internet sources. If you feel that you cannot complete the assignment on your own, seek help from the course staff.

When submitting your assignment, follow the instructions summarized in Section 4 of this document.

Do not use the `break` or `return` statement in any homework or test in CS1132.

1 Population Dynamics

(The following description is taken from Chapter 8.1 of "Mathematical Models in Population Biology and Epidemiology" by Brauer and Catillo-Chavez, 2001.)

Consider a population that is divided into a finite number of age classes labeled from 0 to m. One method of describing the number of members in each age class as a function of time is by using a *linear discrete-time model* for population growth. In such a model, we let $\alpha_{j,n}$ denote the number of members in the j'th class at the n'th time. We assume that the length of time spent in each age class is the same. Then $\alpha_{j,n+1}$, the number of members in the j'th age class at the $(n+1)$st time, is equal to $\alpha_{j-1,n}$ minus the number of members of this age cohort who die before entering the next age class. We assume that the probability of survival from one age class to the next depends only on age. Let p_j be the probability that a member of the j'th age class survives until the $(j+1)$st age class.

All new members recruited into the population are assumed to come from a birth process, with fecundity depending only on age. Assume that there are constants $\beta_0, \beta_1, \ldots, \beta_m$ such that

$$\alpha_{0,n+1} = \beta_0\alpha_{0,n} + \beta_1\alpha_{1,n} + \cdots + \beta_m\alpha_{m,n}.$$

If we define

$$\vec{\alpha}_n = \begin{pmatrix} \alpha_{0,n} \\ \alpha_{1,n} \\ \vdots \\ \alpha_{m,n} \end{pmatrix}$$

and define the *Leslie matrix* to be

$$A = \begin{pmatrix} \beta_0 & \beta_1 & \beta_2 & \ldots & \beta_{m-1} & \beta_m \\ p_0 & 0 & 0 & \ldots & 0 & 0 \\ 0 & p_1 & 0 & \ldots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \ldots & p_{m-1} & 0 \end{pmatrix}$$

then the change in the population through time can be described by the vector difference equation

$$\vec{\alpha}_{n+1} = A\vec{\alpha}_n$$

. In the above equation, $A\vec{\alpha}_n$ is the multiplication of a matrix and a vector, which results in a vector. This operation will be explained below. Let P_n be equal to the total population at time n. Then the vector

$$\frac{\vec{\alpha}_n}{P_n}$$

gives the fraction of the population in each age class at time n.

Goal: To write a script `simulatePopulation` that simulates a population according to a given Leslie matrix and initial population. The simulation has these steps:

1. Define a function `A = getLeslieMatrix(m)` that takes as input an integer m and returns an $(m+1)\times(m+1)$ Leslie matrix with values on the first subdiagonal linearly spaced between [0.9, 0.1] and zeros everywhere else (hint: type in the MATLAB Command Window `help diag` and `help linspace`). These are the survival probabilities. Next, replace the first row of the matrix with a row vector of random integers between [1,5] (hint: use the MATLAB built-in function `randi`; use the `help` command again for an explanation of this function). For example, `A = getLeslieMatrix(5)` might return

$$A = \begin{array}{cccccc} 1 & 5 & 4 & 2 & 1 & 1 \\ 0.9 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.7 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.5 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.1 & 0 \end{array}$$

2. Write a function `w = matrixVectorMult(L,v)`, where L is an $m \times m$ matrix of numbers and v is a vector of numbers of length m. The output w will be a numeric column vector such that `w(i) = L(i,1)*v(1) + L(i,2)*v(2) + ... + L(i,m)*v(m)`. Use SCALAR multiplication operations only, i.e. do not use MATLAB's matrix (vector) multiplication facility.

3. Write the **script** `simulatePopulation` that simulates a population with age-classes $0, 1, \ldots, 10$ (i.e. $m = 10$) for 50 time steps, where the initial population size is 5 individuals in age-class zero. Use your `getLeslieMatrix` function to generate your Leslie matrix for this population. At each time step, use your `matrixVectorMult` function to compute the new population vector, and record the **fractions** of the population in each age-class, i.e. record the vector $\frac{\vec{\alpha}_n}{P_n}$ (hint: you may want to use vector concatenation). Plot the fractions of all ten age-classes through time on the same figure. Add a descriptive title to your plot and label the x-axis as *time* and the y-axis as *Fraction in each age class.*

2 Success-Run Chain

Imagine that you are playing a simple game with m levels, or states. At each level you flip a coin, and if the coin comes up "heads" then you advance to the next level. If, however, the coin comes up "tails", you must go back to the first level and start over. Now imagine that each coin has a different probability of coming up "heads" (in mathematics, this type of random process is called a *success-run chain*). If you were to play this game for an arbitrary length of time, what is the probability that you will be in each of the states?

We can formulate this question more mathematically as follows: let P be an $m \times m$ matrix where the (i, j)th element of P indicates the probability of going **to** i **from** j. For example, if the element in the first row, third column of P is equal to 0.4, this indicates that the probability of going to level 1 from level 3 is 0.4, i.e. the probability of getting a tails in level 3 is 0.4.

Let v_n be a vector of length m, in which the i'th element (where i is between 1 and m) is the probability of being in state i at time n. Multiplying the vector v_n by the matrix P gives the new vector v_{n+1}, which is the probability of being in each state at time $(n + 1)$. Thus, if we wanted to find the long-term probability of being in each state, we would compute v_N for very large N. However, there is a second method we could use to compute the long-term probability distribution: we could simulate hundreds of thousands of trials of the game in which virtual "coins" are flipped according to the probabilities at each state. Each trial would run for some arbitrary number of time steps, and at the end of each trial we record the last state of the player and start a new trial. Amazingly, the empirical distribution of states that emerges from this second method is very close to the computed distribution of states from the first method!

Goal: To write a function `successRunChain()` that demonstrates that the two methods described above (for finding the long-term probability of being in each state) are equivalent. **Note: all subfunctions written for this question must be included in `successRunChain.m`**

1. Write a subfunction `P = getStochMatrix(m)` that takes as input an integer m and returns an $m \times m$ stochastic matrix in which the first $(m-1)$ elements of the top row are random numbers between (0,1) and the m'th element is equal to 1 (hint: use the MATLAB built-in function `rand`). On the first subdiagonal place the complementary values, so that each column sums to 1. For example, `P = getStochmatrix(5)` might return

$$P = \begin{matrix} 0.3 & 0.5 & 0.4 & 0.2 & 1 \\ 0.7 & 0 & 0 & 0 & 0 \\ 0 & 0.5 & 0 & 0 & 0 \\ 0 & 0 & 0.6 & 0 & 0 \\ 0 & 0 & 0 & 0.8 & 0 \end{matrix}$$

2. Write a subfunction `v = simulateCoinsErr(P)` that takes as input a stochastic matrix P, and simulates the success-run chain by repeatedly multiplying the probability distribution by P, using your `matrixVectorMult` function from Question 1 of this assignment. Initialize the probability distribution v_0 so that the probability of being in state 1 is 1. Run the simulation until either 1000 time steps have been taken, or the probability distribution has stabilized such that the change in the distribution from one time step to the next is less than the maximum error of 1×10^{-5}. This second condition can be checked by subtracting the current probability distribution vector from the previous one, taking the absolute value of the vector, and finding the maximum of this vector (hint: use the built-in MATLAB functions `abs` and `max`). If this value, called the error, is less than the maximum error, then we say that the distribution is stationary. Return this stationary distribution as `v`.

3. Write a subfunction `w = simulateCoinsFlips(P)` that takes as input a stochastic matrix P, and simulates the success-run chain by actually performing virtual "coin" flips. Simulate 1×10^6 trials and take 100 steps per trial. Keep track of the state in which each trial ends, and at the end of all 1×10^6 trials, return the empirical distribution of states as `w` (note: remember that a *probability distribution* must sum to one!).

4. In the function `successRunChain()` set the number of levels (states) to be 10, generate a stochastic matrix using `getStochMatrix`, and compute the probability distributions using `simulateCoinsErr` and `simulateCoinsFlips`. Compute the error between these two distributions and print this value to the Command Window (hint: use the built-in MATLAB function `fprintf`).

3 Self-check list

The following is a list of the minimum *necessary* criteria that your assignment must meet in order to be considered *satisfactory*. Failure to satisfy any of these conditions will result in an immediate request to resubmit your assignment. Save yourself and the graders time and effort by going over it before submitting your assignment for the first time.
Note that, although all of these are necessary, meeting all of them might still not be *sufficient* to consider your submission satisfactory. We cannot list everything that could be possibly wrong with any particular assignment!

- Δ Comment your code! If any of your functions is not properly commented, regarding function purpose and input/output arguments, you will be asked to resubmit.
- Δ Suppress all unnecessary output by placing semicolons (;) appropriately. At the same time, make sure that all output that your program intentionally produces is formatted in a user-friendly way.
- Δ Make sure your functions names are *exactly* the ones we have specified, *including* case.

- Δ Check that the number and order of input and output arguments for each of the functions matches exactly the specifications we have given.
- Δ Test each one of your functions independently, whenever possible, or write short scripts to test them.
- Δ Check that your scripts do not crash (i.e., end unexpectedly with an error message) or run into infinite loops. Check this by running each script several times in a row. Before each test run, you should type the commands `clear all; close all;` to delete all variables in the workspace and close all figure windows.

4 Submission instructions

1. Upload files `getLeslieMatrix.m`, `matrixVectorMult.m`, `simulatePopulation.m` and `successRunChain.m` to CMS in the submission area corresponding to Assignment 1 in CMS.
2. Please do not make another submission until you have received and read the grader's comments.
3. Wait for the grader's comments and be patient.
4. Read the grader's comments carefully and think for a while.
5. If you are asked to resubmit, fix all the problems and go back to Step 1! Otherwise you are done with this assignment. Well done!