Assignment 5, Population Modeling

CMPT 102

03-1

1 The Problem

You will be simulating the change in the populations of two species in a given area. The two species have a predator-prey relationship. We'll call the species "predator" and "prey"—you can think of whatever species you like.

A growth model for a predator-prey population was independently suggested by Lotka and Volterra [1]. We use the following variables in the system:

N_1	density of the prey population ($0 \le N_1 \le 1$)
N_2	density of the predatory population ($0 \le N_2 \le 1$)
r_1	rate of increase in of the prey population, in the absence of predation (birth rate) ($r_1 > 1$)
d_2	mortality rate of the predator, absent starvation ($0 < d_2 < 1$)
P	coefficient of predation ($P \ge 0$)
P_2	predatory effectiveness of the predator ($0 \le P_2 \le 1$)

The growth rate in the prey population, N_1 is

$$\tfrac{dN_1}{dt} = r_1 N_1 - P N_1 N_2 \,,$$

and the rate of change in the predator population, N_2 is

$$\tfrac{dN_2}{dt} = P_2 N_1 N_2 - d_2 N_2 \,.$$

Both N_1 and N_2 are the fraction of the total possible population that is present, so it should always be the case that $0 \le N_1 \le 1$ and $0 \le N_2 \le 1$.

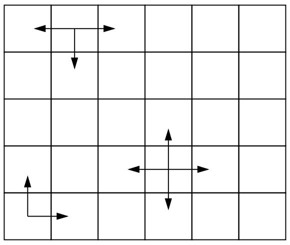
We will simulate the populations of these two species using these formulae.

In order to make things a little more interesting, we will consider the two population in a large area, broken up into a grid. The populations in each area obey the above relationships. In addition to this, some percentage of the population in a square moves to a neighbouring square at each step. We add the following parameters:

m_1 the fraction of the prey population that moves to *each* of the four neighbouring squares with each step ($0 \le m_1 \le 1/4$)

m_2 the fraction of the predator population that moves to *each* of the four neighbouring squares with each step ($0 \le m_2 \le 1/4$)

There should be no migration past the edges of the area. So, for some of the squares, migration happens in less than four directions:



The following parameter settings make for an interesting system. You can also experiment and see if you can create situations that look very different from this one, but still like some real population.

$$\begin{aligned} r_1 &= 1.1 \\ d_2 &= 0.2 \\ P &= 2.0 \\ P_2 &= 0.6 \\ m_1 &= 0.05 \\ m_2 &= 0.05 \end{aligned}$$

2 Program details

You should create a program called `pop.c` to implement this situation over several seasons.

The above formulae give the rate of change in N_1 and N_2. We will be simulating this population in discrete steps. To get from one step to the next, we will set N_1 to $N_1 + \frac{dN_1}{dt}$ and N_2 to $N_2 + \frac{dN_2}{dt}$.

When doing this, you might get one of the N_i either larger than 1 or smaller than 0. You should check at each step and make sure that each N_i

stays in the correct range. If it is larger than 1, set it to 1; if it is less than 0, set it to 0.

After you have calculated the new N_1 and N_2, do the migration. For the prey, $N_1 \cdot m_1$ of the prey should move to the square above the current one; the same number should move to the square below, left and right.

This is trickier than it seems at first glance. You should calculate all of the migration numbers on the original values. That is, if you migrate some individuals south, you should not migrate any of them back north when you calculate migration for that square. That means you probably need another set of storage to hold the numbers of individuals that are migrating. You want to calculate the migrating numbers from the original array, but not change it until you're done these calculations.

You are not required to do any input—all of the parameters (r_1, d_2, etc.) and the initial populations can be set in the code itself. You should make it easy to change these values if you want to explore another type of population, though.

3 The library

In order to get you started and get you over some of the harder parts of the assignment, you should start with the `libpopsim.h` library, provided by the instructor.

Before you `#include` this library, you should define the symbolic constants `WIDTH` and `HEIGHT` which give the size of the grid of squares for the simulation.

So, in your program, you should have something like this:

```
#define WIDTH 10
#define HEIGHT 10
#include "/gfs1/CMPT/102/a5/libpopsim.h"
```

This library contains the following type definition:

```
typedef struct {
  double pred,prey;
} region;
```

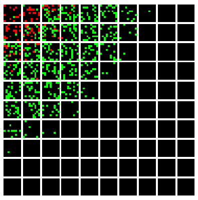
This structure should be used to store the information about a single grid square, with `pred` being the fraction of the maximum predator population present in the area and prey being the fraction of the maximum prey population present.

Also defined is the function draw_img with the following prototype:

```
void draw_img(region pop[WIDTH][HEIGHT], int generation);
```

This function creates an image file called pop00000.ppm in the current directory, where 00000 is replaced with the value generation. The array pop should contain the current populations in each square.

An image like this will be created:



The number of dots in each square represents the number of predators and prey in each location. If several of these images are appropriately combined, they should form an animation of the population dispersal. See Section ?? for information on working with the images.

4 Working with the images

Once you have your program working and creating images, you'll probably want to see what they look like.

4.1 In the lab

Viewing the images is far easier in the lab than from your own computer.

You can view a single image with the command display pop00001.ppm.

You can view all of the images, as frames of an animation with a command like this:

```
animate -delay 10 -pause 2 *.ppm
```

Here, the number after -delay is the 1/100-ths of a second to pause between frames and and number after -pause is the number of seconds to wait before repeating the animation.

If you want to convert the images into an animated GIF file, which can be viewed with a web browser and put on a web page, you can use this command:

```
convert -delay 10 -loop 0 *.ppm popanim.gif
```

This will create a file called `popanim.gif` containing the animation. The number after `-delay` is the 1/100-ths of a second to pause between frames. The `-loop 0` part indicates that the animation should loop after finishing.

4.2 At home

This is going to be a pain. You might be better off to get the program working and then go to the lab to play with the images.

You can transfer the frames of the image to your home computer with FTP and view them in your favourite image viewer. If you want to convert them to another format before transferring them, you can try a command like one of these, depending on the image type you want:

```
convert pop00023.ppm pop00023.bmp
convert pop00023.ppm pop00023.gif
convert pop00023.ppm pop00023.png
```

You can also create and animated GIF as described above and transfer that.

5 Hints

Your `main()` function could get quite large. You might consider breaking it up and creating a couple of functions so it is easier to understand.

You'll probably need to access your array of populations from each of these functions. Since it is a fairly large data structure, you might want to make it a global variable instead of passing it around as a parameter.

- Remember to initialize the values in all of the variables that need it. If you create an array that needs to be initialized, you'll have to write a `for` loop to do that.

- Be careful about the edges of the array. Don't try migrate any individuals from a square in the top row to the square above—there's no square there.
- Suggested plan of attack:
  1. get the program working for a single area;
  2. get the program working on a grid of regions, with no migration
  3. get the image output working;
  4. finally, add migration between regions.

6 References

1. Smith, Robert Leo, **Elements of Ecology**, third edition, Harper-Collins, 1992.

7 Submitting

You have to use the submission server to submit your work. You only need to submit the file `pop.c`. You can do this by typing these commands:

```
tar cvf a5.tar pop.c
gzip a5.tar
```

Then, submit the file `a5.tar.gz`.