Predator Prey Model

Names Here

November 2011

Abstract

Contents

Chapter 1

Introduction

In this task we worked as a group to implement a 2D, sequential predator-prey algorithm with spatial diffusion in Java. We tried to design the program in such a way as to take advantage of Java's object-orientated, modular nature. Due to the large size of our group we also decided to develop a GUI for the program, although the code was designed in such a way as to be usable from the command line as well. ——is it?——

There is always some compromise between readability and performance in coding, and we tried to keep this balance in mind when designing our code. We made best use of class structure whilst still keeping our implementation of the given algorithm as efficient as possible.

The predator prey algorithm implemented in this project crudely models the interaction between population densities of different animals, specifically Hares and Pumas. Each population has a self-interaction coefficient (birth for Hares, death for Pumas) and a coefficient describing how it interacts with other species' populations.

These populations exist on a 'world' consisting of land and water, and are also able to diffuse across land squares with a rate determined by a diffusion coeffient. Thus, each population has N+1 coefficients which determine it's behaviour, where N is the number of animals.

Chapter 2

Model

$H_{ij}^{new} = H_{ij}^{old} + \Delta t(C_1 H_{ij}^{old} + C_2 H_{ij}^{old} P_{ij}^{old} + l(H_{i+1j}^{old} + H_{i-1j}^{old} + H_{ij+1}^{old} + H_{ij-1}^{old} - H_{ij}^{old}))$

for k=1:NumberOfAnimals
if (k=ThisAnimal) then
$N_k^{new} = N_k^{old} + \Delta t C_k(k) N_k^{old}$
else
$N_k^{new} = N_k^{old} + \Delta t C_k(k) N_k^{old} N_{ThisAnimal}^{old}$
end if
end

where

$$C_{Hare} = \begin{pmatrix} H & PH \end{pmatrix}$$

Chapter 3

Design & Implementation

3.1 Code

3.1.1 Structure

3.1.2 Algorithm

3.1.3 Input/Output

3.1.4 GUI

3.2 Tools

3.2.1 SVN

3.2.2 Makefile

3.2.3 Unit Testing

Chapter 4

Performance Analysis

4.1 Testing

Tests were preformed in order to profile the code's performance and scalability. The unix *t*ime command was used to time the code runs, with 'user' and 'system' time being summed to find the total computation time.

The relationship between total computation time per cell and the number of cells was measured in order to quantify the overhead needed by the code (see Figure 4.1). We found that, below a grid size of ∼100 by 100, overhead became important, whereas at larger grid sizes the computation time scaled linearly with the total number of cells. Any scaling worse than linear (such as total computation time being proportional to the $\text{NumberOfCells}^{1.2}$ would be a huge (and unecessary) inefficieny, especially with larger grids.

Some overhead is inevitable in any code, but the amount of effort put into simplifying a problem (such as creating arrays of neighbours) should depend on the expected size of the problem. Our code could probably have had less overhead, allowing it to run faster with smaller gridsizes, but this would likely led to it having a worse (linear) scaling with cell number.

4.2 Analysis

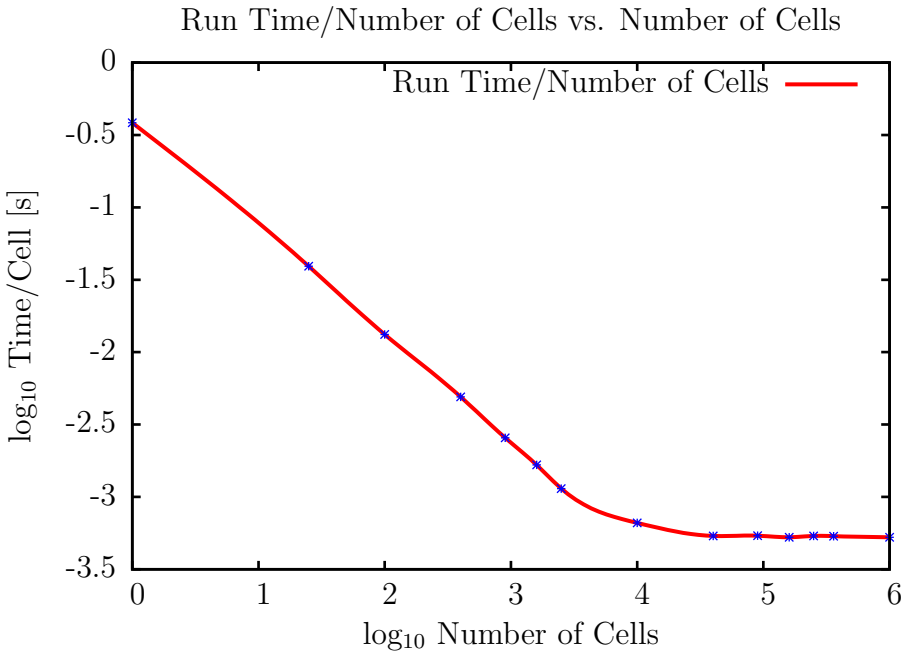

Figure 4.1: This is a graph of total time spent on each cell, which shows that overheads such as array initialization and output (i.e. things not directly involved in the simulation) take up a significant fraction of computation time with grids smaller than $\sim$100 by 100.

Chapter 5

Conclusions

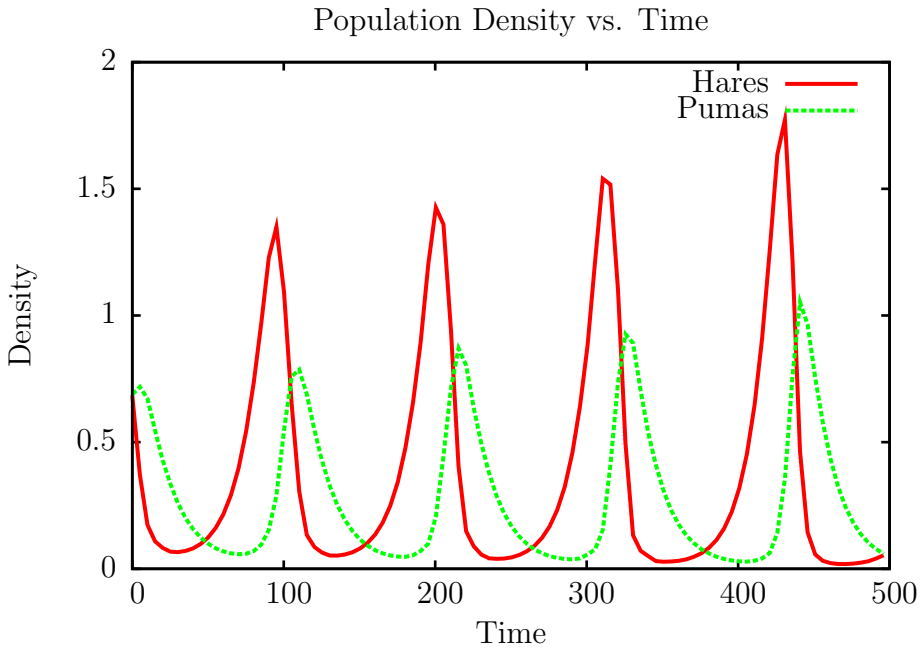

Figure 5.1: Population density vs. time for Hares and Pumas. This periodic behaviour is typical of predator-prey interactions.

Chapter 6

Group Evaluation