MSc in High Performance Computing

MSc in High Performance Computing with Data Science

Programming Skills

Development Coursework Assignment

Adrian Jackson, Mark Bull, David Henty, Alan Simpson and Mike Jackson

23 August 2018

1 Introduction

In this coursework you will implement a sequential (not parallel) version of a two-dimensional predator-prey model with spatial diffusion using one of Fortran, C, C++, Java or Python. The aim is to produce a "best practice" scientific code within a full development framework (i.e. revision control, build tools, unit test framework etc.).

The coursework is a group exercise where you will collaborate with your fellow students in small groups to implement the model, write tests and write user and developer documentation. The groups are randomly assigned by the course organiser, based on both the programming languages you are familiar with and want to use.

2 Modelling pumas and hares in a landscape

You are going to model the interaction of pumas (predator) and hares (prey) in a two-dimensional landscape. Pumas eat hares. Without hares, pumas decline and die out through lack of food. Without pumas, hares breed and increase in numbers forever. Both animals tend to spread out (or diffuse) to fill parts of the landscape where there are fewer of their own kind. Their landscape contains areas of water where neither pumas nor hares can live.

Partial differential equations can be used to model the behaviour of pumas and hares within a landscape:

$$\frac{\partial H}{\partial t} = rH - aHP + k(\frac{\partial^2 H}{\partial x^2} + \frac{\partial^2 H}{\partial y^2})$$

$$\frac{\partial P}{\partial t} = bHP - mP + l(\frac{\partial^2 P}{\partial x^2} + \frac{\partial^2 P}{\partial y^2})$$

where:

- H is the density of hares (prey).
- *P* is the density of pumas (predators).
- *r* is the birth rate of hares.
- *a* is the predation rate at which pumas eat hares.
- *b* is the birth rate of pumas per one hare eaten.
- *m* is the puma mortality rate.
- *k* is the diffusion rates for hares.
- *l* is the diffusion rates for pumas.

Do not worry if you don't understand these equations, as discrete versions that can be used in your program will be introduced shortly. What the equations say is that:

- The change in hares over time is affected by the birth rate of hares, the rate at which hares are killed by pumas and the diffusion of hares through the landscape.
- The change in pumas over time is affected by the birth rate of pumas, which also depends on the hares available as food, the natural death rate of pumas, and the diffusion of pumas through the landscape.

2.1 Numerical approximation

The mathematical model above treats time and space as continuous quantities but continuous quantities cannot be handled directly by a computer. We need to use a discrete approximation. To do this, we can divide space and time up into discrete units. The landscape can be modelled as a rectangular grid, of dimension $N_x \times N_y$, where N_x is the number of columns and N_y is the number of rows, and where each grid square can be either land or water. Within each grid square there will be a number of hares, H_{xy}, and pumas, P_{xy}, living in it, where H_{xy} and P_{xy} are >= 0.

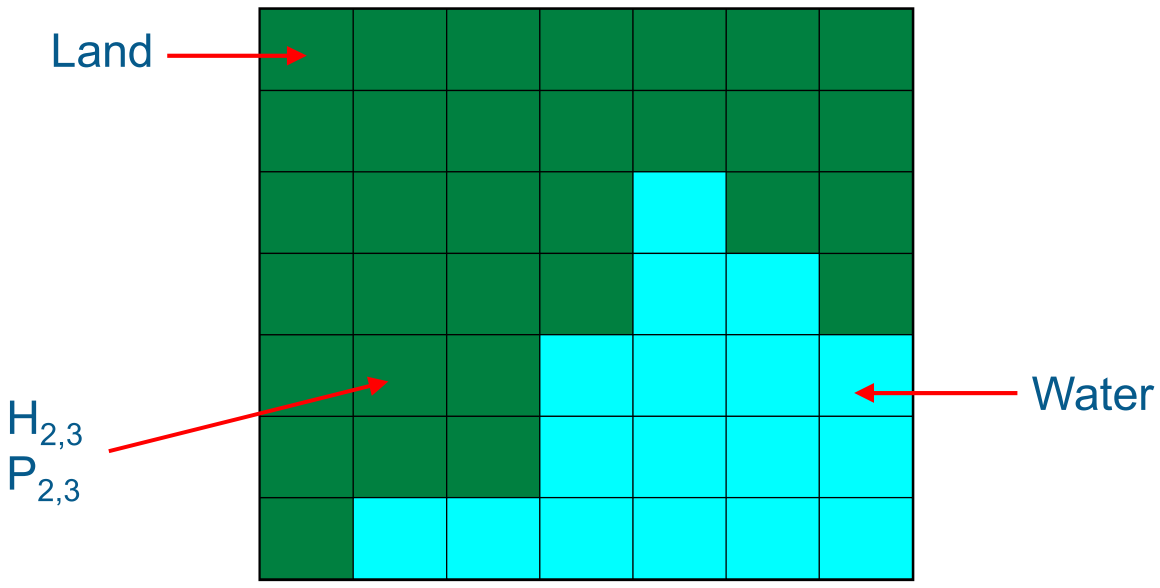

Figure 1: 2-dimensional landscape grid. Squares are indexed by column then by row from the bottom-left.

Time can be modelled by successive, discrete, time-steps, of size Δt. The numbers of hares and pumas in the landscape at next time step depend on the numbers present in the current time-step. So, the new number of hares, H, at time $t + \Delta t$, in a specific grid square, is given by:

$$H_{ij}^{new} = H_{ij}^{old} + \Delta t(rH_{ij}^{old} - aH_{ij}^{old}P_{ij}^{old} + k\left(\left(H_{i-1j}^{old} + H_{i+1j}^{old} + H_{ij-}^{old} + H_{ij+1}^{old}\right) - N_{ij}H_{ij}^{old}\right))$$

The number of hares in a grid square at the next time step equals the number of hares there at present adjusted by the change in population of hares over the timestep. The population change is a combination of the number of hares born in that square (where *r* is the hare birth rate), the number of hares killed in that square (which depends on number of pumas in the square, where *a* is the puma predation rate) and the number of hares that move into or out of that square from the neighbours (where k is the diffusion/migration rate).

N_{ij} is the number of "dry" (i.e. non-water) grid squares out of the four squares neighbouring the square *(i, j)* (to the north, south, east and west). From this approximation, we assume that hares cannot move diagonally nor can they swim.

The new number of pumas, *P*, at time *t +Δt*, in a specific grid square, is given by:

$$P_{ij}^{new} = P_{ij}^{old} + \Delta t(bH_{ij}^{old}P_{ij}^{old} - mP_{ij}^{old} + l\left(\left(P_{i-1j}^{old} + P_{i+1j}^{old} + P_{ij-1}^{old} + P_{ij+1}^{old}\right) - N_{ij}P_{ij}^{old}\right))$$

The number of pumas in a grid square at the next time step equals the number of pumas there at present adjusted by the change in population of pumas over the timestep. The population change is a combination of the number of pumas that die in that square (where *m* is the puma death rate), the number of pumas born in that square (which depends on number of hares in the square where *b* is the puma birth rate) and the number of pumas that move into or out of that square from the neighbours (where *l* is the diffusion/migration rate). Pumas, like hares, cannot move diagonally, nor can they swim.

We can assume that the distance between grid squares is 1 i.e. Δx = Δy = 1, as scaling can be absorbed into the diffusion terms, *k* and *l*.

2.2 More on the birth and death of hares and pumas

Let's look at the birth of hares:

$$H_{ij}^{old} + \Delta t(rH_{ij}^{old} \dots)$$

As an example, let's take the basic unit of time as 1 month, and let's assume that every pair of hares breeds 3 times a year (once every 4 months) and each produces a litter of 5 baby rabbits (called leverets). So, in 4 months, 2 hares produce 5 more hares, so each hare produces 0.625 hares a month, so the hare birth rate, *r*, is 0.625. To check that this is correct, let's substitute into our equation: $H_{ij}^{new} = 2 + 4(0.625 \times 2)$ which equals 7 (the 2 parents and the 5 offspring).

As we could have thousands of hares breeding continuously, we can use a much smaller value of *Δt*. A value for *r* can depend upon the units chosen e.g. months, days etc. Likewise, it can improve the scalability of our model if, instead of recording the hare, and puma, populations as integers, we record them as real numbers in terms of densities per grid square e.g. 3.53 hares per hectare.

As for the death of hares:

$$H_{ij}^{old} + \Delta t(rH_{ij}^{old} - aH_{ij}^{old}P_{ij}^{old} \dots)$$

Hares only die through being eaten by pumas, not naturally, so more pumas means more hares are killed. More hares also means more are killed as they are easier to find.

Looking at the birth of pumas:

$$P_{ij}^{old} + \Delta t(bH_{ij}^{old}P_{ij}^{old} \dots)$$

Unlike hares, the birth rate of pumas (*b*) depends not only on the number of their own kind but also on their available food, the hares.

As for the death of pumas:

$$P_{ij}^{old} + \Delta t(bH_{ij}^{old}P_{ij}^{old} - mP_{ij}^{old} \dots)$$

Pumas differ slightly in that, unlike hares, they have no predators, so die at a natural rate (*m*).

2.3 More on movement of hares and pumas across the landscape

We could just model a single population over the whole landscape, but this is not very realistic. We want a finer-grained model which allows for different densities of pumas and hares at different points in the landscape. Our model will be more realistic if we model how animals move around the landscape. For hares, we have a diffusion term, *k*:

$$k(\frac{\partial^2 H}{\partial x^2} + \frac{\partial^2 H}{\partial y^2})$$

In the discrete form of our equation, this is represented as a “random walk”, defined as:

$$k\left(\left(H_{i-1j}^{old} + H_{i+1j}^{old} + H_{ij-1}^{old} + H_{ij+1}^{old}\right) - N_{ij}H_{ij}^{old}\right)$$

The number of hares in surrounding squares is summed. From this is subtracted the number of hares in the current grid square multiplied by the number of surrounding land-only grid squares (N_{ij}). The total value is then multiplied by a hare diffusion term, *k*.

For pumas a similar calculation is used, with a puma diffusion term, *l*.

3 Starting your coursework

Groups of between 3-4 students are randomly assigned by the course organiser, based on both the programming languages you are familiar with and want to use.

You should get together in your group and create your design and list of tasks. The tasks are up to your group and do not have to be single-person tasks, you are free to work together as you think best. You should ensure that the overall amount of work each member of your group does is equally divided.

Here are some examples of tasks that could be suggested (although not all of these may be appropriate for your group's design and task breakdown).

- Set up and maintain a source code repository (e.g. Git or Subversion).
- Create the build environment.
- Define tests.
- Implement the computational kernel.
- Implement the I/O part of the code.

- Implement the test framework.
- Write documentation.

Note that you are not required to use this list or any of the items on it for your tasks. It is provided as an example of the sorts of tasks you could consider.

Once the tasks have been defined you should assign them amongst the group members to ensure that each person does (roughly) equal amounts of work. As development proceeds it may be necessary to change the tasks or re-assign work to ensure that people have the correct amount of work to do. It will also be beneficial to assign tasks to people's skills wherever possible.

You should ensure that everyone has some coding/development and some documentation work.

Any conflicts that cannot be resolved within the group will be mediated by the course organiser, on request.

4 Source code requirements

You should use good programming practice such as comments, whitespace and indentation, function descriptions and definitions, etc. You are strongly encouraged to put your code and tests under revision control, use a build tool (e.g. Make or ANT) and implement tests since you will be marked on these.

It is expected that all groups will get a working implementation.

4.1 Landscapes

Your code must be able to handle landscapes up to 2000 × 2000 grid squares ($1 <= N_x <= 2000$ and $1 <= N_y <= 2000$).

Your code must be able to cope with arbitrary “water” grid squares, where hare and puma population values are always zero.

The hare and puma densities can be modelled with two-dimensional double precision floating point arrays. C programmers are strongly encouraged to use either statically declared arrays, or the `arralloc()` routine[1]. This routine's (`.h` and `.c` files) are in the LEARN area for this coursework (see Appendix – coursework assignment files on LEARN). Alternatively, 1D arrays could be used. C++, Java and Python programmers may consider more object-oriented designs. Different designs have different strengths and weaknesses

The boundary conditions are $H = P = 0$ on the edges of the landscape. This can be implemented by adding a halo of grid squares around the edges of the landscape and making them all “water”.

[1] We provide `arralloc` to simplify the process of allocating memory in C as it generally causes a lot of problems for people and wastes a lot of time unnecessarily. `arralloc` wraps up all the complexities of `malloc` and provides a nice clean interface for creating arrays. However, if you are confident with your C and are happy to do your own memory allocation then feel free to do so.

4.2 Inputs

Your code must be able to read in a bitmask ASCII file describing which points are land or water. The format of this file consists of one line giving N_x, the number of columns, and N_y, the number of rows, followed by a sequence of space separated ones and zeros (land=1, water=0). For example:

```
7 7

1 1 1 1 1 1 1

1 1 1 1 1 1 1

1 1 1 1 0 1 1

1 1 1 1 0 0 1

1 1 1 0 0 0 0

1 1 1 0 0 0 0

1 0 0 0 0 0 0
```

The LEARN area for this coursework (see Appendix – coursework assignment files on LEARN) has examples. See Appendix - how to create your own landscape for how you can generate your own land/water mask files.

The user must be able to set all the parameters in the equations. These are as follows, with suggested default values in brackets:

- *r* is the birth rate of hares (0.08).
- *a* is the predation rate at which pumas eat hares. (0.04)
- *b* is the birth rate of pumas per one hare eaten (0.02).
- *m* is the puma mortality rate (0.06).
- *k* is the diffusion rates for hares (0.2).
- *l* is the diffusion rates for pumas (0.2).
- *Δt*, the size of the time step (0.4).

In addition, the user must also be able to set *T*, the number of time steps between outputs (see below).

You can use random initial densities of *H* and *P*, say in the range 0 to 5.0, or you could find out what happens when a large group of pumas are placed in a confined area, with a uniform distribution of hares.

Run the simulation from time *t*=0 to *t*=500.

4.3 Outputs

Your program must output:

- A “Plain PPM” file every *T* timesteps, where *T* is the number of time steps between outputs. “Plain PPM” is a simple uncompressed ASCII format (like PGM but colour). Run `man ppm` for details.
- Average values of *H* and *P* over the grid at regular intervals. These averages can be calculated across the grid as a whole, or just across land-only grid squares. How you output this (whether you put the values into a file or just print them to standard output) is up to you. The frequency of output is up to

you, depending on how you wish to interpret “regular intervals” but the number of intervals must be greater than 1.
- When the simulation completes, the total time that was taken to execute the model simulation. Again, how you output this is up to you.

4.4 Equations

Implement the discrete approximations of the equations, as they are defined above.

Using more accurate approximations of the time-derivatives and of the equations - using optimisations of the equations themselves derived numerical analysis techniques - should *not* be attempted. This is a programming skills exercise not a numerical analysis one, and an important aspect of programming skills is meeting the requirements of the customer, which, in this case is implementing the equations as defined above.

The implementation should be sequential, this is *not* a parallel programming exercise.

4.5 User interface

Your program must be runnable from the command-line in batch mode. In this context, "batch mode" means once the user presses RETURN your program should run the simulation to completion and output the results without prompting the user for further information or action.

Whether you also support other interface approaches e.g. an interactive command-line interface or a GUI is up to you.

4.6 Tests

Testing is important for any code, so you need to ensure that you can validate the results of the code you produce, especially if you make changes to fix bugs or optimisations.

Using a test framework can help to automate the running of your tests and reporting of their results.

4.7 Execution environment

Please note that your code and tests must satisfy the following:

- It must be possible to build and run your code and tests on either Cirrus or DICE.
- It must be possible to build and run your code and tests from the command-line.
- If your code or tests requires additional packages or software to build or test that are not already available on Cirrus or DICE then you should document what these packages or software are, where to get them, and how to set them up.

Non-working or non-compiling code will be marked down accordingly.

4.8 Documentation

You must also provide documentation, which must include:

- Information on the programming language, revision control, debuggers, build tools, and test tools you have used.

- Whether your code runs on Cirrus or DICE.
- Where to get, and how to build and install, any third-party packages needed by your code (for packages that are not already on Cirrus or DICE).
- How to build your code.
- How to run your code.
- How to run your tests.
- Summary of key design decisions and reasons for these.

5 Submission and marking

You are required to submit your source code and to complete a WebPA (Web Peer Assessment) contribution assessment[2].

5.1 Submitting your source code and documentation

Source code and documentation submissions are done via the Turnitin submission tool from within LEARN.

Submit a single archive, `zip` or `tar.gz`, file containing all your source code, build scripts, test code and documentation.

The filename of your submission must include your *exam number* which is the B number which appears on your matriculation card (this is not the same as your student number). You should also include the course identifier, `PS` for Programming Skills, and the identifier `Dev` for this coursework. So, for example, if your exam number is `B123456`, then you would name your submission file `B123456-PS-Dev.zip` or `B123456-PS-Dev.tar.gz`.

While every member of your group needs to submit a copy of the source code so that we can return marks to all students via Turnitin and LEARN, only one of these (per group) will be marked. The one to be marked will be picked from a member of the group at random, so make sure you all upload *exactly the same copy of the source code archive*.

5.2 Completing a WebPA contribution assessment

The WebPA contribution assessment is accessed via LEARN. It is completed online. You must complete this individually. The assessment covers categories of contribution relating to the coursework. For each category, you have 100 points to assign across the members of your group, including yourself.

A more equal distribution of points in a category means that in your view all members made a similar level of contribution or contributed equally to that category. If, in your view, a group member contributed more to a specific category, you can award them more points to reflect this.

The categories are:

1. **Designing the implementation.**
2. **Implementing the design and tests.**
3. **Writing the documentation.**

[2] https://www.ed.ac.uk/information-services/learning-technology/assessment/webpa

4. **Collaborating:** How well did the group member collaborate during the coursework? Did they propose constructive ideas and suggest alternatives options? Did they attend meetings? Did they arrive on time? Did they reply to emails in a timely way? Did they carry out their designated tasks? Did they keep to your group's deadlines? Did they let you know of any problems as soon as possible?

5.3 Deadline

The deadline for submission for your source code, and completion of the WebPA contribution assessment, is **16:00 Wednesday 7th November (week 8) 2018** for all students.

Submission are allowed up to 7 calendar days after the deadline, with a 5% deduction per day or part-day late.

If you submit your source code by the deadline then it will be marked, and you cannot submit an amended version to be marked later on.

If you do not wish the submitted version of your source code to be marked, and intend submitting an amended version after the deadline, then it is your responsibility to let the course organiser know, *before the deadline*, that the submitted version should not be marked.

5.4 Marking

Source code and documentation will be marked out of 100 as follows:

- Source code:
  - Meeting requirements: 20
  - Quality of design: 20
  - Presentation and readability: 20
  - Quality of tests: 20
- Documentation: 20
- **Total: 100**

Individual marks will then be adjusted in light of the WebPA contribution assessments. The maximum this can vary the mark for an individual student (without MSc course board intervention) is plus or minus ten marks (+/- 10). All marks are subject to moderation by the MSc course board.

Groups are expected to attempt to resolve any issues early, requesting the involvement of the course organiser if necessary, so a significantly negative mark modification should not be made without good reason. A group which assigns a significantly negative modifier to one member, without having previously contacted the course organiser with concerns about that one member, may be penalised for failing to do so.

The mark for this assessment forms 70% of your overall mark for the Programming Skills course.

5.5 Provisional marks

Your provisional marks and feedback will be returned no later than 16:00 Wednesday 28th November 2018. All marks are provisional until confirmed by the MSc course board.

6 Questions

Feel free to e-mail the course organiser if you have any questions. Questions of interest to the whole class will be anonymised and them, plus their answer, forwarded to the whole class.

If problems occur in how a group is collaborating, the group should initially attempt to mediate and resolve the problems amongst themselves. Should this prove to be insufficient, issues should be directed to the course organiser. Likewise, any conflicts that cannot be resolved within the group will be mediated by the course organiser, on request.

There follow answers to some common questions...

6.1 Can we use GitHub or BitBucket?

Yes. BitBucket may be preferable as it allows you to create a private Git repository you can share amongst your team without allowing others to view it.

6.2 Can .PPM files and screen dumps be provided as evidence that we've met the requirements of the practical?

Yes, but these will be considered as supplementary evidence. The code itself is the primary evidence that you have met the requirements.

6.3 Can we use someone else's algorithm to calculate random numbers (so long as we put in a reference for it)?

Yes. Some languages provide built-in libraries for random number calculation, some do not. If your chosen language does not then feel free to use a third-party algorithm, library or package. Make sure you provide a reference for it. You should also check that its licence allows you to use it (open source licencing is covered in Software Development next semester).

6.4 Other questions

- What should our code do if the population density becomes negative?
- How should the location of the landscape file be provided to the program?
- How robust should our program be in terms of valid or invalid user input values?
- How should we visualise the population of hares and pumas across the landscape at a specific point in time, using “Plain PPM” files?
- How do we get around the “Plain PPM” restriction that “No line should be longer than 70 characters”?
- Is it OK to use global variables?

This are all design decisions you need to make. There are many options available. Choosing an appropriate solution is one of the challenges of this coursework.

7 Appendix – coursework assignment files on LEARN

To access the LEARN area for this coursework:

- Log-in to LEARN at http://www.learn.ed.ac.uk OR via http://www.myed.ed.ac.uk, using your EASE username and password.
- Click My Courses.
- Click Programming Skills (2018-2019)[SV1-SEM1].
- Click Assessment.

- Click Assignment 1: Development

To download a file, right-click on the associated link and select Save As...

8 Appendix - how to create your own landscape

You can create your own landscape via a number of approaches.

8.1 Write by hand using a text editor

You can create landscape files in a text editor.

8.2 Write a simple program

You can write a simple program to create landscape files with specific dimensions and with specific distributions of land and water.

8.3 Use `image_to_dat`

The LEARN area for this coursework (see Appendix – coursework assignment files on LEARN above) contains a Python program, `image_to_dat.py`, that will convert image files to landscape files. This Python program will run under any Python deployment in which numpy and the Python Image Library (PIL) have been installed (for example any Anaconda distribution of Python).

Using any image editor or paint program, create a bitmap (`bmp`) image with your landscape, using black to represent land and white to represent water.

Run `image_to_dat.py` as follows, specifying your image file and the name of the file you want your landscape to be called. For example:

```
$ python image_to_dat.py landscape.bmp landscape.dat
```

The LEARN area has example landscapes you can try e.g.

```
$ python image_to_dat.py corner.bmp corner.dat

$ python image_to_dat.py random.bmp random.dat
```

8.4 Use `gimp` (DICE)

If you have access to the image manipulation program `gimp` (which is available on DICE but not Cirrus), then you can use that in conjunction with a `pnm2dat` executable that is available (along with its source code) in the LEARN area for this coursework (see Appendix – coursework assignment files on LEARN above). This converts PNM files, produced by `gimp` to ASCII files.

Start the image manipulation program `gimp`.

Select **File => New...**.

Choose the height and width (assuming one grid square = one pixel).

1. Click **Advanced Options**.
2. Select **Colour space: Greyscale**.
3. Click **OK**.
4. Draw your map.
5. Select **File => Export As...**
6. Enter a filename with an ending `.pnm` e.g. `file.pnm`
7. Click **Select File Type**
8. Select **PNM image**
9. **Export**.
10. Select Data Formatting: Ascii
11. Click **Export**.
12. Select **File->Quit** to exit
13. Run `./pnm2dat file.pnm file.dat` to convert the PNM file to the same format as the supplied input file (recall that `./` tells bash that the file to be executed is in the current directory). Dark points (greyscale < 128) are considered land (represented as 1 in the `.dat` file, light points (greyscale >= 128) are water (represented as 0 in the `.dat` file).

To change brush size in `gimp`, select **Windows => Dockable Dialogs => Tool Options**.