

Project 1: A Model for the Spread of Rabies

Rabies is a fatal disease affecting a number of animal species, some domestic and some wild. In rare cases it may be contracted by humans, normally through a bite from a rabid dog. In many places it is endemic, but in other countries such as Australia and Britain, it is not currently present, and there are strong efforts being made to keep rabies out. To this end, a number of mathematical models have been developed for the spread of the disease. This project investigates one of these models, and examines the effectiveness of methods by which the spread of the disease could be reduced or prevented. The model is taken from Chapter 20 of “Mathematical Biology”, by J.D. Murray (Springer, 1989, 1st edition; a third edition (2002) in two volumes has also appeared in which the model is in Volume 2 Chapter 13). Copies of the most relevant pages will be given out, and the general features of this kind of modelling will be discussed in lectures.

By far the most significant carriers of rabies during European outbreaks are foxes, so a reasonable approximation for the spread of the disease is to consider the population dynamics of just these animals. (In the US the culprits are raccoons.) Let S be the density per square kilometer of susceptible foxes, ie those that have not yet caught the disease but are liable to do so, and let I be the corresponding density of those that are infectious carriers of the disease. After non-dimensionalisation, the governing equations we will adopt are

$$\frac{\partial S}{\partial t} = -IS + bS(1 - S)$$

and

$$\frac{\partial I}{\partial t} = IS - \lambda I + \frac{\partial^2 I}{\partial x^2};$$

these equations will be explained and justified in lectures. They depend on only one space variable x and so model features such as plane waves spreading parallel to the boundaries across a country which is infinite in one direction and finite in the perpendicular (x) direction. We will suppose the country is isolated geographically (foxes cannot swim, at least not that far), so that boundary conditions $\partial S/\partial x = 0, \partial I/\partial x = 0$ will be adopted at both boundaries (corresponding to zero fox fluxes there).

1. Write a program to solve the partial differential equations given above by stepping forward in time. The lectures will tell you three methods to use for the timestepping. Two of these feature estimates of the derivatives which are second order, centred in space and time, and with a timestep set by the user. The third uses MATLAB's **ode45** integrator which employs a variable timestep to

achieve a user-specified accuracy (or its MATLAB default value). Two of the methods will turn out to work well, and one has some interesting problems.

For this first exercise, program up the 2nd order Adams-Bashforth method; this will be explained in lectures. Your program should have the following parts (at least!)

- (i) a section which reads in input parameters, does any necessary housekeeping operations, sets the timestep and any timestep-dependent constants, specifies the frequency at which results will be output, and generally gets everything ready for the integration in time to proceed
- (ii) a simple timestepping algorithm, which is used solely to get the calculation started by taking one Euler or RK2 step
- (iii) an accurate (second order in time) timestepping procedure, which uses Adams-Bashforth in the way described in the lectures. Both this and the previous routine need properly to incorporate the boundary conditions
- (iv) a section which outputs and/or plots the results at some specified times. This may also produce an animation; we will supply you with a function **animate.m** which does this in MATLAB.

Depending on both your own programming style and the language you are using, some of these tasks may be assigned to subroutines or functions. The problem needs quite a few input parameters, and you may choose to hard-wire these into the program (which must then be recompiled before each run), or you may prefer to read them from an input file. Suggested parameters to read from such a file are the length of the computational domain, quantities to specify the initial conditions, λ, b , a factor to set the timestep, the length of time to continue the integration, and the time interval between dumps to the plot file. A sample input file will be handed out.

Run the program and perform any obvious debugging. In the first instance, check the resolution necessary for your code to give accurate answers, by doubling or halving the mesh size. Aim to get the plots looking imperceptibly different at high enough resolution. In your write-up, include plots showing results at three different resolutions, one inadequate and two satisfactory. Check that your code gives qualitatively similar behaviour to that found in Murray. Note it is impossible to reproduce the graphs there exactly without knowing details of the initial conditions, which are not given—this is naughty of Murray!

2. Now perform more quantitative checks as follows. For suitable values of λ , measure the wave speed in your solutions and compare them with the theoretical prediction on pp 677-678 in Murray (2002), making sure you can get satisfactory agreement. In your write-up, state clearly everything you did to achieve this. Analytically, show that there is a steady state solution which is spatially uniform, with $S = \lambda, I = b(1 - \lambda)$, and by examining the growth or decay of linearised perturbations proportional to $e^{(st+ikx)}$ show that this solution is stable (this procedure will be explained in more detail in lectures). Then run your program for long enough to check that your solutions reach this steady state with the correct values for S and I . Include description and plots for this in your write-up.

3. Next, make a new version of your program which attempts to do the timestepping by a different method known as second order leapfrog (this will be explained in lectures). You will need to add some extra arrays and change the subroutine which does the regular timestepping. Take one of the cases you computed earlier and compare how well each method does. Present plots showing the comparison, making sure they are for identical times. Identify any discrepancies, and check whether using higher resolution or a smaller time step makes any difference. Which method do you feel is better, and why?
4. Finally, use Matlab's inbuilt 4th/5th order Runge-Kutta solver **ode45** to solve the problem. (If you have not seen this before, use Matlab's help to discover how it works). You will need to create a routine to evaluate the right hand sides of the two equations. This should have as input arguments the instantaneous time, and the current values of the susceptibles and infectives arranged as a 1-D vector of length $2n$, where n is the number of spatial grid points including end points. The susceptibles are stored in the first n entries, and the infectives in entries $n + 1$ to $2n$. The routine needs to return a column vector of length $2n$ containing the right hand sides, first of the susceptibles at meshpoints 1 to n , and then, in positions $n + 1$ to $2n$, of the infectives. Because **ode45** will wrongly update the infectives at the boundary points, these two values must be reset using the boundary conditions at the start of the RHS routine (see lectures).

Once **ode45** has done the integration, it will have created a 2-D array with one suffix corresponding to the n susceptibles followed by the n infectives, and the other suffix corresponding to the time values at which you asked for results. However, it will have chosen its own timestep to proceed from one time value to the next. (Use of a variable step integrator to achieve a specified accuracy, rather than prescribing a fixed timestep at the outset, is usually computationally more efficient.) You will need to decide how to plot your results; as stated earlier we will be supplying a routine **animate.m** on the course website which will enable you to display movies of the results, but in your write-up you must select individual plots. Again, you need to remember that the infectives will have been wrongly updated at the boundaries, so before plotting, you will need to get inside the results array and reset the boundary point infectives using the boundary conditions (the interior points should all be correct). This needs to be done for each time level.

Take one of the cases you calculated earlier and present results obtained with the new method, comparing against the leapfrog and Adams-Bashforth methods.

In your conclusion, include an assessment of how you rate the three methods against one another. Pick your favourite and attempt to reproduce the results in Figure 13.7 of Murray's book.