D Walker d.c.walker@sheffield.ac.uk

Ecolab – Agent based Predator-prey simulation in Matlab

1. Description of model

This Matlab based programme simulates a simple predator-prey system consisting of interacting populations of foxes and rabbits. Initial populations sizes can be selected by the user and are randomly distributed in a square 'environment', (dimensions=km, size=user defined) within which there is a random distribution of vegetation.

1.1 Agent memory and functions

The memory parameters of both foxes and rabbits are:

age - age of agent in number of iterations
food - current food content (arbitrary units)
pos - current position in Cartesian co-ords [x y]
speed - number of kilometres agent can migrate in 1 day (also gives food search radius)
last_breed - number of iterations since agent last reproduced.

The model is run iteratively (1 iteration=1 day – fixed) and for each agent in turn, the following rule set is applied:

eat – describes how agents eat vegetation (or each other!)
migrate – describes how agents move around in the environment
die – determines how agents are killed off
breed – determines how agents reproduce

The current rules can be summarised as:

Eat – specified in functions @fox/eat.m, @rabbit/eat.m

SUMMARY OF FOX EAT RULE
1. Fox calculates distance to all surviving rabbits
2. Fox identifies nearest rabbits(s) within its search radius (same as speed)
3. If more than one equidistant, one is randomly picked
4. Probability of fox killing rabbit depends on distance of rabbit and speed of fox
5. If probability > random number, fox moves to rabbit location and rabbit is killed
6. If fox does not kill rabbit, it's food is decremented by one unit

NOTE If a fox kills rabbit agent ***m*** , the rabbit does not die immediately. A one will be placed in MESSAGE.dead(***m***) and it will 'commit suicide' at the end of the iteration.

SUMMARY OF RABBIT EAT RULE

1. Rabbit detects food level in its 1km x 1km square of the environment
2. If food> 1, rabbit will consume food, otherwise rabbit food level decremented by one unit.

Migrate – specified in functions @fox/migrate.m, @rabbit/migrate.m
SUMMARY OF FOX MIGRATE RULE

1. If a fox has not eaten, it will pick a random migration direction
2. If it will leave the edge of the model, the direction is incremented by 45 degrees and it will try again (up to 8 times)

SUMMARY OF RABBIT MIGRATE RULE

1. Rabbits will migrate only if they have not eaten in the current time step
2. Rabbits will always try to migrate towards the nearest food source
3. The rabbit will extract the distribution of food in its LOCAL environment (at distances < its daily migration limit)
4. It will identify the location of the nearest food and migrate into it.
5. It's new position will be randomly placed within this square
6. If no food is detected within its search radius it will move randomly

***Die* – specified in functions @fox/die.m, @rabbit/die.m**
Foxes and rabbits die if their food level reaches zero or they are older than *max_age*
If agent ***m*** dies, it does not disappear immediately, but a '1' (one) will be placed in MESSAGES.dead(***m***) and it will 'commit suicide' at the end of the iteration.

Note – in order to preserve the numbering of agents, dead agents are replaced with an empty data structure {}

***Breed* – specified in functions @fox/breed.m, @rabbit/breed.m**
Foxes and rabbits can reproduce if their current food level is above a threshold level, and there has been sufficient time elapsed since the last time reproduced. Both agents and foxes produce a single offspring when they reproduce.

Note - reproduction is asexual (there is no concept of gender) and instantaneous (there is no concept of pregnancy).

All the parameters which control agent behaviour are created by a function called *create_param_file* which is called at the start of every simulation. The parameters are stored in a data structure called PARAM which is saved in a data file and reloaded.

1.2 Environment

The model environment is represented by a data structure containing the following fields:

ENV_DATA.shape - shape of environment - FIXED AS SQUARE
ENV_DATA.units - FIXED AS KM
ENV_DATA.bm_size - length of environment edge in km

ENV_DATA.food is a bm_size x bm_size array containing distribution of food

Initially food is a bm_size x bm_size array, with each square containing 50 units of food. Values in a 20 x 20 square are then set to zero. This is to simulate an area of drought where there is no vegetation.

You can access the level of food in a particular square (e.g. xp, yp) using the following:

food=ENV_DATA.food(xp, yp);

so if, for example the food distribution at a particular point in time is as shown here: :

$$\begin{array}{c|cccc|c} & \multicolumn{4}{c}{(y) - km} & \\ & 1 & 2 & 3 & \ldots & \\ \hline & 5 & 0 & 100 & \ldots & 1 \\ & 3 & 57 & 6 & \ldots & 2 \\ & 54 & 47 & 98 & \ldots & 3 \\ & \vdots & \vdots & \vdots & \vdots & \vdots \\ \end{array} \quad (x) - km$$

and xp=2km and yp=3km, then

ENV_DATA.food(xp, yp)=6;

1.3 Communication and timing

The global data structure MESSAGES contains the following fields:

MESSAGES.atype - n x 1 array listing the type of each agent in the model (1=rabbit, 2-fox, 0=dead agent)
MESSAGES.pos - list of every agent position in [x y]. Dead agents have the 'dummy' position [-1 -1] (this is never accessed).
MESSAGES.dead - n x1 array containing ones for agents that have died in the current iteration

The model is run synchronously i.e. message lists are only updated at the end of each iteration (in update_messages).

1.4 Randomisation

Matlab's random number generator is used to generate the initial locations of agents, and also within some functions (e.g. in migrate.m, in order to make decisions about which direction to migrate in). This means that if you run two simulations with the same numbers of agents, you may still get different results.

In order to override this, if you set the input into the random_selection function to be zero (*ecolab.m* line 41), the same order of random numbers will be generated as for the previous simulation, and you should get the same results for the same initial agent number and environment size.

Note – randomisation of agent order is NOT implemented in this model

1.5 Simulation control

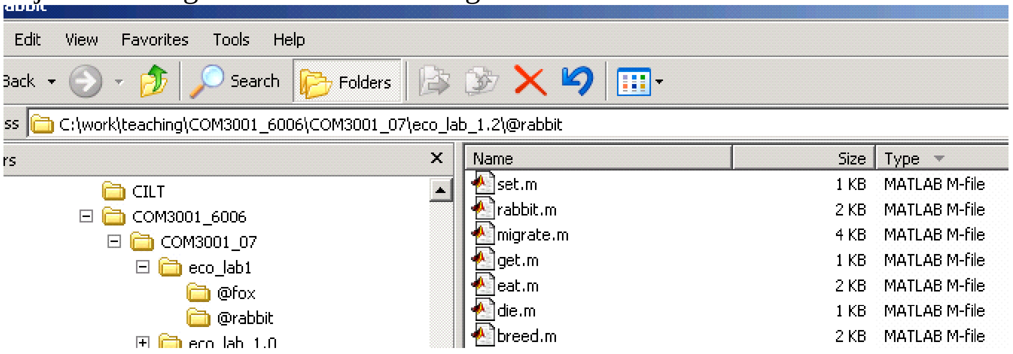
The global structure CONTROL_DATA contains data to enable "fast mode" to operate and to allow a pause flag to exist. This mode aims to speed up operation by reducing the amount of plotting that occurs as the number of live agents increases.

If *fast mode* is set to "true" it is also used to stop the program if the number of rabbits or foxes reaches zero.

CONTROL_DATA.fmode_display_every – a value describing the iteration interval between plotting if the *fmode* variable is set to "true".
CONTROL_DATA.fmode_control – a 2x5 array containing agent no. thresholds and associated iteration intervals to be used to set *CONTROL_DATA.fmode_display_every*
CONTROL_DATA.pause – a Boolean flag which is used to enable pausing within a *while* loop

2 File/directory structure

The programme files are under a directory called *eco_lab*. The two class declaration directories (for foxes and rabbits) are beneath this main directory:
Inside each class directory is the class declaration function (i.e. fox.m or rabbit.m, the object set and get functions and the agent 'rule' functions:

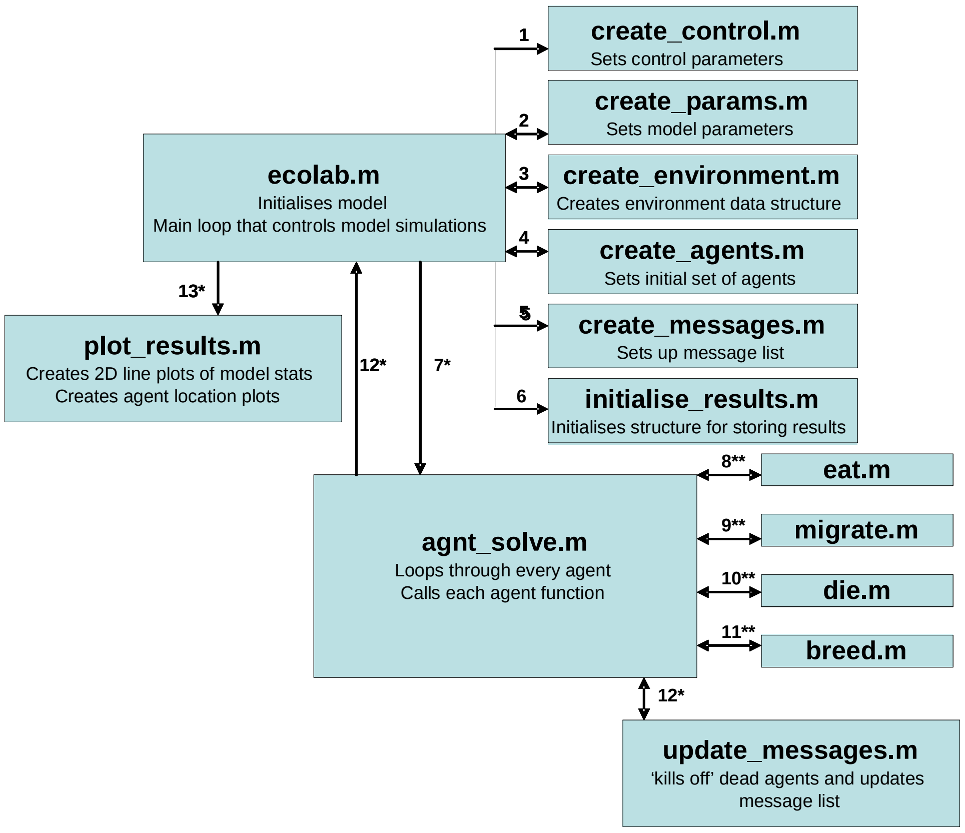


C:\work\teaching\COM3001_6006\COM3001_07\eco_lab_1.2\@rabbit

Name	Size	Type
set.m	1 KB	MATLAB M-file
rabbit.m	2 KB	MATLAB M-file
migrate.m	4 KB	MATLAB M-file
get.m	1 KB	MATLAB M-file
eat.m	2 KB	MATLAB M-file
die.m	1 KB	MATLAB M-file
breed.m	2 KB	MATLAB M-file

3. Data Flow

Data flow in the model, including the sequence of all major function calls, is shown below:


create_control.m
Sets control parameters
create_params.m
Sets model parameters
ecolab.m
Initialises model
Main loop that controls model simulations
create_environment.m
Creates environment data structure
create_agents.m
Sets initial set of agents
create_messages.m
Sets up message list
initialise_results.m
Initialises structure for storing results
plot_results.m
Creates 2D line plots of model stats
Creates agent location plots
agnt_solve.m
Loops through every agent
Calls each agent function
eat.m
migrate.m
die.m
breed.m
update_messages.m
'kills off' dead agents and updates message list
1
2
3
4
5
6
7*
8**
9**
10**
11**
12*
12*
13*

* - function is called from within time step loop
** - function is called from within agent loop (which is inside time step loop)

4 Running the model

Copy the zip file onto to your PC. Unzip the file.
Open Matlab and change the working directory to *eco_lab*

The function you need to call to run the model is called *ecolab*, with inputs in the following form:

ecolab(size, nr, nf, nsteps, fmode(optional),outImages(optional))

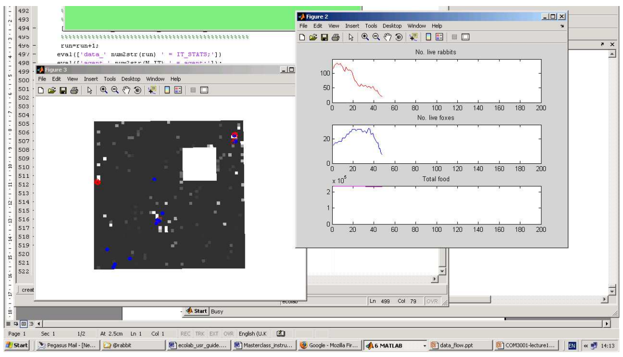
where:

size is the size of the model environment in km.
nr is the initial number of rabbits
nf is the initial number of foxes

nsteps is the total number of iterations to be executed.

(optional inputs)
fmode is a Boolean (true/false) to turn on or off a “fast mode” (default: true=on)
outImages is a Boolean (true/false) to enable image output (on=much slower operation, default: false=off)

Two new figures will appear. Figure 2 will slow line plots of total rabbit number, total fox number and total remaining food on separate axes. Figure 3 will show the distribution of rabbits (red circles and foxes (blue circles). The dark background represents the distribution of food (white= no food). The latter figure is updated only once every 5 iterations as plotting can slow down the simulation fairly significantly. If you which to change the frequency at which this plot is produced, you need to edit *plot_results.m* line 61.



5 <u>Saving and plotting data</u>

While each simulation is running, all the data relating to that simulation will be stored in a global data structure called IT_STATS:

IT_STATS=struct('div_r',[],...	no. rabbit births per iteration
'div_f',[],...	no. fox births per iteration
'died_r',[],...	no. rabbits dying per iteration
'died_f',[],...	no. foxes dying per iteration
'eaten',[],...	no. rabbits eaten per iteration
'mig',[],...	no. agents migrating per iteration
'tot',[],...	total no. agents in model per iteration
'tot_r',{[]},...	total no. rabbits in model per iteration
'tot_f',{[]},...	total no. foxes in model per iteration
'tfood',[]);	total amount of food remaining

At the end of each simulation, these statistics will be saved in a file called results_nr_***nr*_*nf***.mat where ***nr*** is the original rabbit number, and ***nf*** the original fox number. You can load these results into the Matlab command workspace and manipulate them yourself to create line plots showing multiple results sets, e.g..

```
>>load my_file1 IT_STATS
>>figure
>>plot(IT_STATS.tot,'b-')
>>hold on
>>load my_file2 IT_STATS
>>plot(IT_STATS.tot,'r-')
etc.
```

Note that if you repeat a simulation with the same initial numbers of agents, then your existing results files will be overwritten, so you may want to rename the files to prevent this happening.

Saving and copying figures

You can save Matlab figures by selecting file-→save in the figure window. Saved figures can be re-opened by typing:

```
>>open my_fig.fig
```

on the command line (where *my_fig* is the name you used to save the figure), or by double clicking on the file in the file manager window.

You can copy figures directly into Windows applications such as Powerpoint by selecting edit → copy figure in the figure window, then pasting in the other application.

Automatically saving Images and Creating Movies

If the ecolab optional input parameter ***outputImages*** is set to true, the application is set up to save the agent plot (figure 3) as a set of jpeg images Note: If "fast mode" is also set to ***on*** (which it is by default) then only a subset of images will be output. To output all image set fmode=false and imageOut=true – this will be slow!

If you wish, you can then try and use a freeware movie creator (e.g. videomach) to turn these images into and AVI. I have **not** attempted this on University networked machines, so I cannot guarantee this will work!

Note: every time you run the model, the jpeg images will be overwritten. Also, saving images takes up memory (and slows down the simulation a lot -especially writing to you networked user space on managed PCs). You might also want to delete these after you have finished the exercise to stop filling up your file space quota!