University of Strathclyde Engineering

ME108 Engineering Analysis and Numerical Methods

Graded Assessment: **Coursework 1 (2025)**

INTRODUCTION

This assignment covers all the topics learned in Semester 1, and applies them to **computational modelling and simulation**, a fundamental skill in engineering analysis. You will design and implement a MATLAB function that simulates the evolution of a simplified wolf–rabbit ecosystem. This coursework involves **array manipulation, loops, conditional logic, and function design**, which are essential for numerical methods and real-world engineering applications.

BACKGROUND

In this coursework, these concepts are simplified into a **grid-based model**, where each cell represents a patch of land that can contain either:

- **2 = Wolf (predator)**
- **1 = Rabbit (prey)**
- **0 = No animals (empty)**

The grid evolves over time according to a set of rules that mimic survival, reproduction, and death. This approach is similar to **cellular automata**, which are used in various fields such as physics, biology, and computer science.

The best known of these is the *Game of Life*, created by mathematician John Conway in 1970, is a cellular automaton that simulates how simple rules can lead to complex behaviour. It takes place on a grid of cells, each of which can be alive or dead. The state of each cell evolves over discrete time steps based on the number of live neighbors it has—cells survive, die, or are born according to fixed rules. Despite its simplicity, the Game of Life produces fascinating, often unpredictable patterns that can grow, move, or stabilize over time. It has become a classic example in mathematics, computer science, and complexity theory, illustrating how order and chaos can emerge from minimal initial conditions.

SYSTEM REPRESENTATION

- The ecosystem is represented as a **square 2D array** or matrix in MATLAB.
- Each cell interacts with its **8 neighbouring cells** (including diagonals).
- To avoid edge effects, the grid uses **wrap-around boundaries** (toroidal topology), meaning the top connects to the bottom and the left connects to the right.

EVOLUTION RULES

The state of each cell in the next generation depends on its immediate neighbours:

1. **Prey Overpopulation**: A rabbit will die if more than 3 neighbouring cells also contain rabbits.
2. **Prey Underpopulation**: A rabbit dies if fewer than 2 neighbouring cells contain rabbits.
3. **Re-population**: An empty cell becomes a rabbit if at least 3 neighbouring cells contain rabbits.
4. **Predation**: A rabbit dies if at least one wolf is a neighbour.
5. **Predator Starvation**: A wolf dies if no rabbit is in any of its surrounding cells.

YOUR TASK

Write a MATLAB function that implements the evolution of the system given the initial state and the number of generations. The function should be of the form:

```
function [E_evol, n_wolf, n_rabbit] = rabbitwolf(E0, ngen)
% Inputs:
% E0:  Initial 2D array representing the ecosystem.
% ngen:  Number of generations to simulate.
%
% Outputs:
% E_evol:  Final evolved grid.
% n_rabbit:  Number of rabbits remaining.
% n_wolf:  Number of wolfs remaining.
```

Input checks: Your function needs to check that the inputs provided are valid for the function. Specifically, it should verify:

- `E0` must be square
- `E0` contain only a wolf ($= 2$), or rabbit ($= 1$), or is empty ($= 0$).
- `ngen` must be positive (`ngen` ≥ 1).

If any check fails, your function should:

- Return a matrix of zeros for `E_evol` that is the same size as E0. *Hint*: `zeros(n)` returns an n-by-n matrix of zeros.
- Return `n_rabbit = 0`, `n_wolf = 0`.
- Display a warning message. *Hint*: there are several commands to display text to the command window, such as `disp`, `warning`, or `fprintf` which is a carry-over from c/c++.

SUBMISSION REQUIREMENTS

- Submit a **single MATLAB file** named `rabbitwolf.m` via MyPlace. There is a template for the function available on MyPlace.
- Include a header with your name and registration number:

```
%% ME108 Assignment 1 2025 Solution File
% Submitted by: [Your Name], Reg. No. [Your Student Number]
```

MARKING SCHEME

A series of tests using different inputs are performed to assess the performance of your code. The marking is based on only on the outputs your function returns, not on the elegance or clarity of your implementation.

20 marks Function definition and input validation.

30 marks Correct representation of neighboring cells.

30 marks Implementation of survival and death rules.

20 marks Iterative evolution process.

Important: If your function fails the first function validation test (available on MyPlace), your grade will be **zero**.

Function validation test

A validation script is provided named `validator.m`. Please save the validator script in the same folder as your assignment script and run from another script or the command line.

```
>> validator
==================================================
This is just a validation test of your submission.
It DOES NOT check the correctness of your answers.
It just check that your file is valid for submission.
==================================================
CORRECTLY SAVING THE OUTPUT E_evol: the size of E0 and E_evol matches
CORRECTLY SAVING THE OUTPUT n_rabbit: the number of rabbits are >=0
CORRECTLY SAVING THE OUTPUT n_wolf: the number of wolves are >=0
```

If your fails the validation, it is awarded a zero mark (0).

Operation and evolution test sets

Further sets of tests are then performed to check: correct form of the (user) inputs and outputs, correct implementation of the death/survival rules and neighbour extraction, correct implementation of the evolution scheme. If your function passes all tests correctly it is awarded a perfect score (100%).

Function definition, correctness of inputs/outputs This checks first that the function passes validation, which checks the outputs are in the correct form, the function accepts the correct number of inputs, etc. Then it checks against the rules on the inputs. For example, that the matrix E0 is square, that E0 can only contain valid numerical values (e.g., 0 = empty, 1=rabbit, 2=wolf), and the ngen is an positive integer. The test also ensures that if an input is found invalid, that the outputs are returned as 0 (or a matrix of zeros).

Representation of the surrounding cells For ngen = 1, this set of tests look at the correct identification of the ground, rabbit and wolf, and then the correct identification of surrounding cells, including cases in the middle, then those that 'wrap around' the sides, corners, etc. Note: the test matrices for M are all equal to or larger than 4×4.

Death/survival rules implementation This tests all the rules surrounding the evolution rules, e.g., rabbit dies from overpopulation, empty cells become rabbits (new rabbits are 'born'), rabbits die from wolf kills, wolves dies from starvation.

Iterative evolution process This tested the evolution of sets of randomly generated scenarios changing the size of M, and size of ngen and compares the output against the reference function.