Lab - Week 3

Markov Chains and System Dynamics Models

1. Warm-up: Simple Markov Chain examples

Here is a very simple (but perpetually annoying) problem: Socks get lost all the time. There are always single ones without any match! Can you predict which proportion of your socks will normally be available in properly matched pairs?

A sock can be in four different states: paired (P), unpaired (U), worn (W), lost (L). For the purpose of this exercise, do not worry about the fact that pairs actually require two socks. We handle each sock as a separate entity. You wear two socks (one pair) each day and we model each day as one time step. Each sock has a probability $p_{wp} = 0.2$ to be worn on any given day if it comes from a matched pair. Sometimes, out of pure despair, you wear unpaired socks. This happens with probability
so, wash does not need to be a state, because it will certainly be washed
$p_{wu} = 0.02$. At the end of the day you put the worn socks into the washing. When they come out of the washing they usually end up paired again (with $p_p = 0.8$). You may not find the matching sock so that they end up unpaired (with $p_u = 0.1$), or you loose the sock somewhere in the washing with $p_l = 0.1$. Lost sock have a tendency to sometimes turn up again and to be found. This is an even chance $p_f = 0.5$. When this happens they initially become an unpaired sock. Finally, every now and then, you find the match for an unpaired sock by chance with $p_m = 0.05$.

Tasks

1.1 Draw the state transition network and use this to determine how many fresh pairs of socks you will have available on any given day in the long run equilibrium. Before you do this, convince yourself that this long-run equilibrium must exist. You can use *Mathematica* or *Python* with *Numpy* to solve this.

1.2 What would change if a lost sock would never be found again ($p_f = 0$)?

1.3 [optional] Propose an interesting problem/scenario of your own to which you could apply Markov Chains in the same way. Have your suggested problem checked by a tutor (it may not be suitable for modelling as a Markov Chain), then solve it.

2. System Dynamics - ODE Models (Ant Foraging)

We discussed in the lecture how mass-foraging ants, such as Argentine ants, communicate and collectively learn about their environment by using pheromones (chemical markers). When they are foraging, they are trying to select the best food source. Best can mean many different things: best food quality, shortest path to the food source, easiest path to the food source, etc. Here we focus on the length of the path. To make this selection as a group (and finally to achieve consensus on the selection), the ants employ a form of collective reinforcement learning that is mediated by pheromones. We consider an experiment in which two paths of different lengths lead to from the nest to two different food sources. On their way out from the nest foragers select the left or the right branch with a probability that depends on the relative amount of pheromone they sense on this branch. On their way back from the food source each forager deposits new pheromone on the path she uses. The amount of this is modulated by the path length (the shorter the path, the more reinforcement, ie. the higher the pheromone deposit). This creates a positive feedback loop. Pheromone, being a chemical, also evaporates over time, which moderates the positive feedback with a negative feedback loop. Let’s explore how this learning dynamics plays out...

Tasks

2.1 Implement and visualise a differential equation system (ODE) system that describes the foraging behaviour.

Denoting the pheromone level on branch *i* with c_i, we can describe the dynamics of the pheromones as

$$\frac{d\,c_i}{d\,t} = \frac{1}{l_i}\,\frac{f(c_i(t))}{f\,(c_1(t)) + f\,(c_2(t))} - \rho \cdot c_i\,(t)$$

$-\rho \cdot c_i\,(t)$ is the term that comes from the evaporation, $\frac{f(c_i(t))}{f\,(c_1(t)) + f\,(c_2(t))}$ is the proportion of foragers choosing branch *i* depending on the relative amount of pheromone on it and $1/l_i$ is the amount of pheromone deposited on this branch (ie. the path quality is equivalent to the inverse of the length; short paths are good).

The strictly monotone non-linear function *f* weighs the importance of pheromones. We use

$f\,(x) = (k + x)^{\mu}$

with $k = 0.2$, $\mu = 2.1$

In the following, we use the evaporation constant $\rho = 0.05$ and the path lengths

$l_1 = 2$
$l_2 = 2.2$

To solve the system we also need to set initial conditions. We set

$c_1\,(0) = c_2\,(0) = 1$

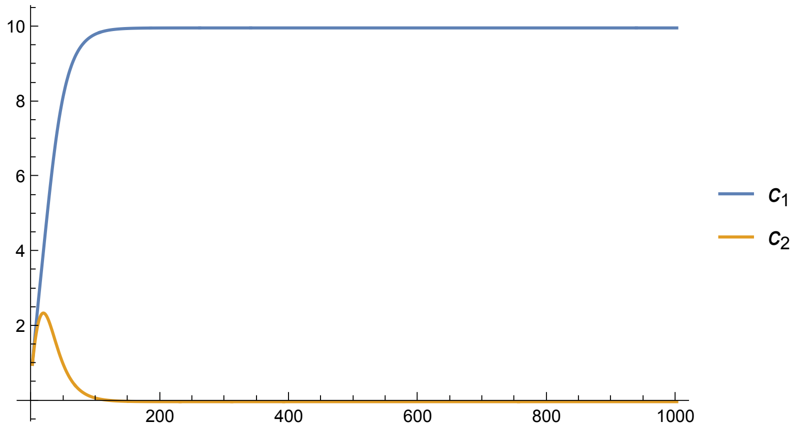
There is no particular reason why we would need to choose 1 here.

If you wish to have a look at this in a “Box and Diagrams” System Dynamics notation, you can start from the following scenario on insightmaker.com: https://insightmaker.com/insight/4IbyCcL2-KaP3kp9qV70E7b

We now solve the system numerically in *Python*. To do so, we use the library *SciPy* which provides a function *odeint* for the forward integration of ordinary differential equation. We have provided a separate notebook “*ODE example.ipynb*” to demonstrate its use.

When you have successfully implemented and visualised this system you should see something like the following plot.

Out[ ]=



2.2 Adding uncertainty

As an introduction to the next lecture, let us think about what would happen if we let the system converge in a first phase of the experiment and then add a second phase in which we switch the lengths of the branches around. For example, for the first phase use the parameters ρ=0.05, k=0, μ=2, l1=2, l2=3.

At t=1000 all traffic on branch 2 will well and truly have disappeared. If the path lengths are now switched around, can the system recover and adapt to this? Reason about this abstractly before you proceed.

Verify your hypothesis computationally simply by setting the initial conditions of the simulation for Phase two to the final result that you obtained for Phase 1. You could, of course, also explicitly build the two phases into the simulation. This would be best done by turning l_1 and l_2 from constants into functions of time and adapting the code accordingly.

The answer is, of course, that the system cannot adapt to this change. At the end of Phase 1 all traffic is on Branch 1 so that the (now shorter and better) Branch 2 does not get a chance to recover since it cannot collect pheromone! There are no foragers left on it!

Modifying the simulation

In reality the branches are, of course, never completely empty. Ants will sometimes just make random decisions and not depend 100% on the pheromone information. Will the system dynamics change if we account for this?

Above we have set k=0. This ensures that the pheromone determines completely (!) how many ants go one one branch. We can set k>0 for a cheap-and-dirty way to account for the randomness. Effectively, this models that there is always a residual probability for ants to go on a branch even if there is no pheromone at all on it.

Will this change the system behaviour? Reason about this question abstractly and verify with the simulation.

Warning: adding the constant *k* is, of course, not truly adding a treatment of randomness to the system. It is just a crutch to simply capture the relevant effect. Systems with randomness (so-called stochastic systems) can behave fundamentally different from deterministic approximations and we need to use true representations of randomness to account for this. This is easily done in simulations by adding some noise and you can try this out with the code above. It is a bit tricker mathematically and we will dive into this in the next lecture.