Lab 3 part 1 - Simulating Ashby's Homeostat

Return to main labs page: **Main Labs Page (https://canvas.sussex.ac.uk/courses/34985/pages/main-labs-page-3)**

The *Sandbox* framework code, and all other code for labs 1, 2 and 3 is here:

> **AS_autumn_2025_exercises.zip (https://canvas.sussex.ac.uk/courses/34985/files/6130167?wrap=1) ↓ (https://canvas.sussex.ac.uk/courses/34985/files/6130167/download?download_frd=1)**

Ashby's Homeostat - an ultrastable machine (system)

Ashby used his Homeostat machine to demonstrate his theory of ultrastability. His machine was composed of 4 units, each of which was ultrastable in its own right. As all of the units in the Homeostat are ultrastable, and they only connect to each other, then it follows that the Homeostat is also ultrastable. The Homeostat is ultrastable in a *self-adaptive* way, which makes it a very good example for us, but it is also possible for a system to be ultrastable by *modifying its environment*, either instead of or as well as modifying itself.

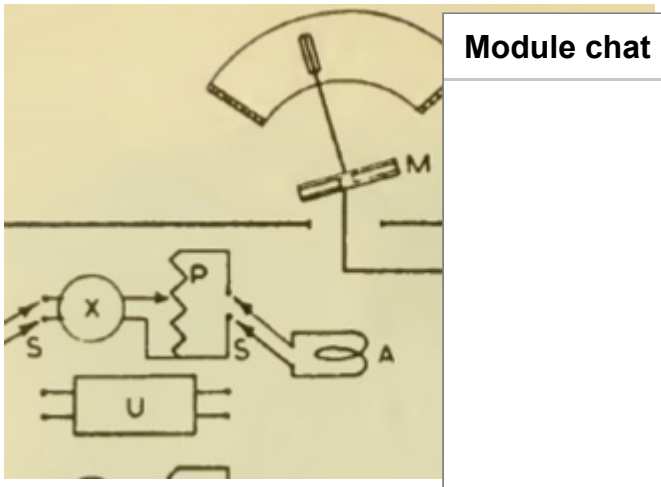

Figure 1. The Homeostat's essential variable is represented b

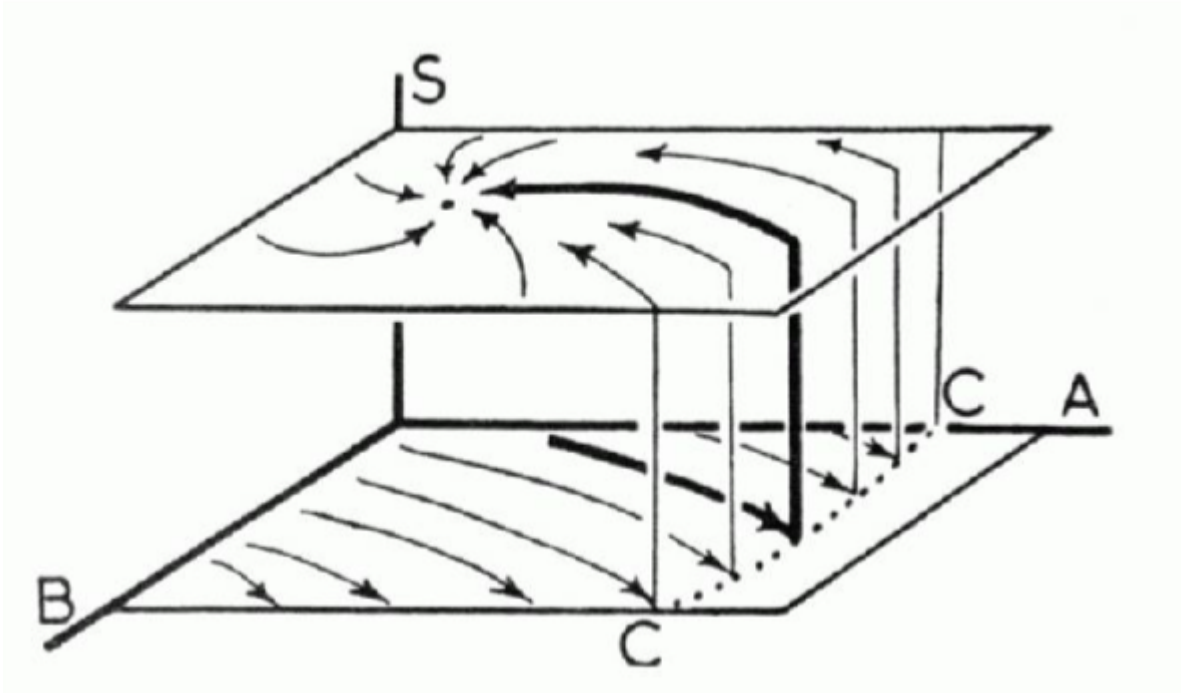

Figure 2. When an ultrastable system changes its parameters, its field in phase space (i.e. its lines of behaviour) changes.

As we saw in the lecture, a Homeostat unit is designed to keep its *essential variable*, represented by the angle of a rotating stylus, or needle (**Figure 1**), within *viable limits*. As long as the essential variable is within those limits, the unit's input weights don't change. On the other hand, if the essential variable leaves the viable region, then the unit begins to modify its input weights. Every time a unit modifies its weights, the unit waits for some period of time to see whether the new weights bring the essential variable back into the viable region. If they do not, then the weights are changed again, and then after the same period of time the essential variable is checked once more, and this process is repeated again and again until the essential variable is back within the viable region. Every time the weights are changed, a new field is selected in the phase space of the essential variable (a plane in the dimensions of θ and $\dot{\theta}$). If we take **Figure 2** as an illustration of the phase plane of a single Homeostat unit, then each plane (and all of the many other possible planes which are not shown) corresponds to a different set of input weights. As long as the fields are unstable, as in the lower of the two illustrated planes, the weights will continue to change. When the field selected by the current weights causes the essential variable to be moved to a stable fixed-point *within the viable region*, then the weights will stop changing. To emphasise this important last point: viability implies some kind of stability, but the converse is not necessarily true - a system may be stable, but not viable (e.g. if the system is in the stable state of being broken, or even dead) even if all of the lines of behaviour in the field converge on a single fixed point, the ultrastable system will only stabilise if that point lies within the viable region. Another point which is not illustrated in **Figure 2**: as long as the essential variable lies within the viable region, it doesn't *have to* stabilise on a fixed point - it could be continuously varying (changing) according to some fixed pattern, or even varying randomly or chaotically (note that these last two kinds of change are *not* the same thing).

This is the essence of ultrastability - if the ultrastable system is already in a viable state, then it just carries on with the existing *field*, which is made up of the *lines of behaviour* which the system

can follow through its state space. If the system is not in a viable state, then it changes its parameters in order to select lines of behaviour which lead the system back to viability. When the parameters of a system are changed, it may not be immediately clear whether or not the correct field has been selected - this is why Ashby's Homeostat waits for a while after changing its weights before checking again whether the system is in a viable state - it can take some time for a system to stabilise even after its parameters are set correctly. This is also why, when we use an `adapt_fun` with a *Sandbox* `Controller`, the controller also waits for a period of time set by its `test_interval` parameter.

We will see more examples of this kind of scenario in *Sandbox* in later labs. When simulating a *self-adaptive* mobile agent in Sandbox, we can use a `DisturbanceSource` to implement disturbances which have a negative effect on the behaviour of the agent, and we can use the `adapt_fun` function of an agent's controller to change the controller's parameters (e.g. weights) when the agent is disturbed away from stable lines of behaviour. As we shall see later on this page, the simulated Homeostat also has an `adapt_fun` function.

The simulation

I created a *Sandbox* simulation of Ashby's Homeostat for two main reasons: firstly, the theory of ultrastability is a very important one, and my hope is that having a simulation of an ultrastable machine to examine in this lab will help you to understand it. Secondly, Homeostat-related projects are a very common choice for the second assignment for this module, so by providing a working simulation of the system, I can save students the time it would take to create their own (which could take anything from hours to days) so that they can spend that time on the more important aspects of their projects.

In the second part of this lab, we will look at how to use a `DisturbanceSource` with a `Homeostat`, but in this first part we will just always start our simulation with the Homeostat in a disturbed, i.e. non-viable, state. This will mean that the Homeostat will always begin to adapt at the beginning of the simulation, and we can use the time that the Homeostat takes to reach viability to compare the effects of the different parameters and techniques that we will use for the Homeostat.

Our main script for today is `lab3_part1.py`.

Your experiments & explorations

Number of Homeostat units

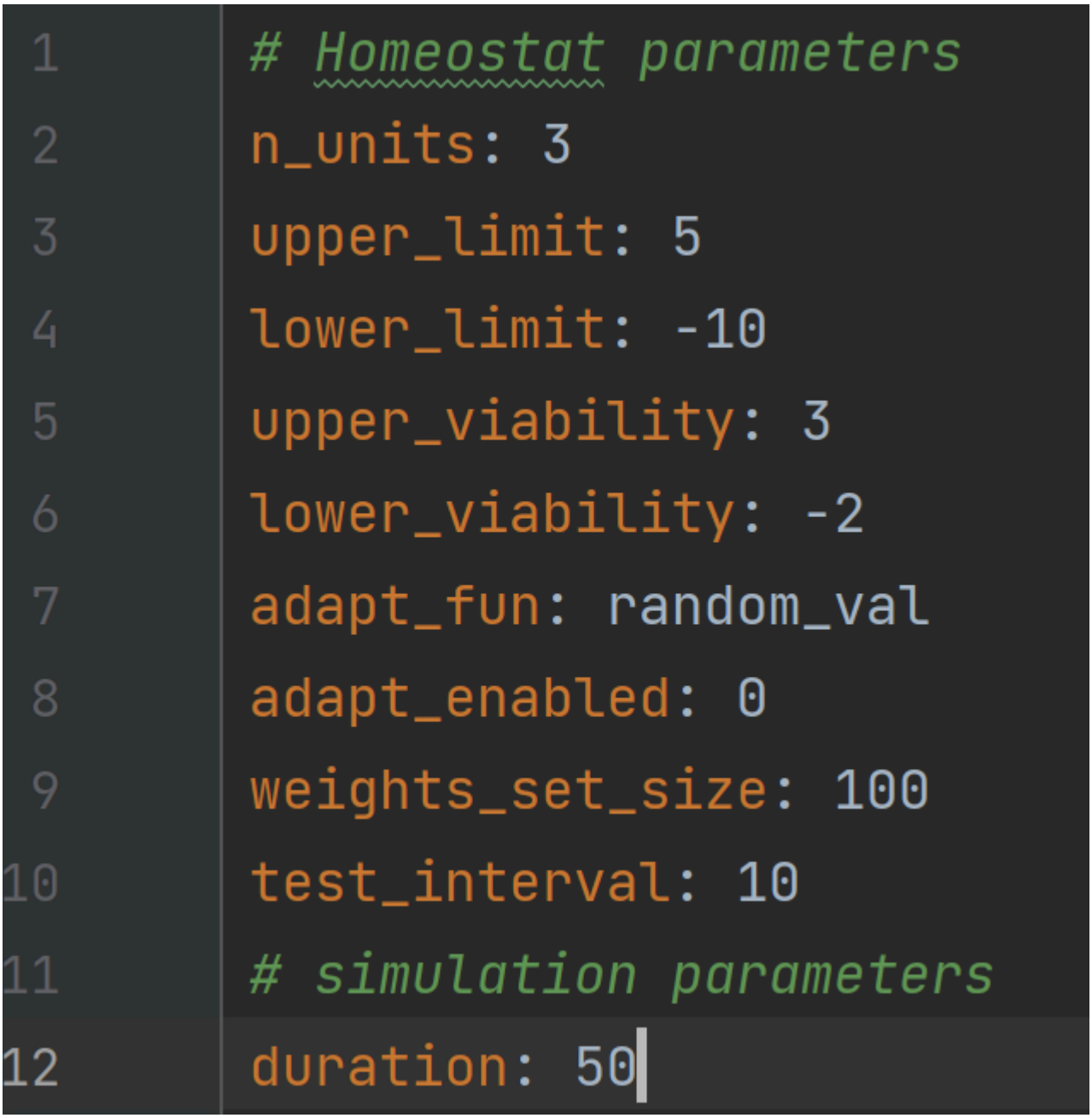

Figure 3. The parameters that you will experiment with in this class, in the `params.yaml` file.

As long as all of the units in the Homeostat are fully interconnected, the time it takes for the Homeostat to adapt will be strongly (but nonlinearly) correlated with the number of units. This is set by the n_units parameter (**Figure 3**).

Tasks

1. Try running the simulation with 1 unit, 2 units, 3 units, and so on. You will eventually reach the point where you will need to increase the simulation duration because the system will take longer to stabilise (`duration`, **Figure 3**). If you keep going, you will also eventually find a number of units which will be too large for adaptation to take place in a reasonable time.
    - Does it look like the time taken to stabilise scales linearly or nonlinearly with the number of units? (we'll take a look at how to quantify this kind of thing later)
    - What is the largest number of units which you can obtain good results for, in a short space of time?

The Homeostat's `adapt_fun` function

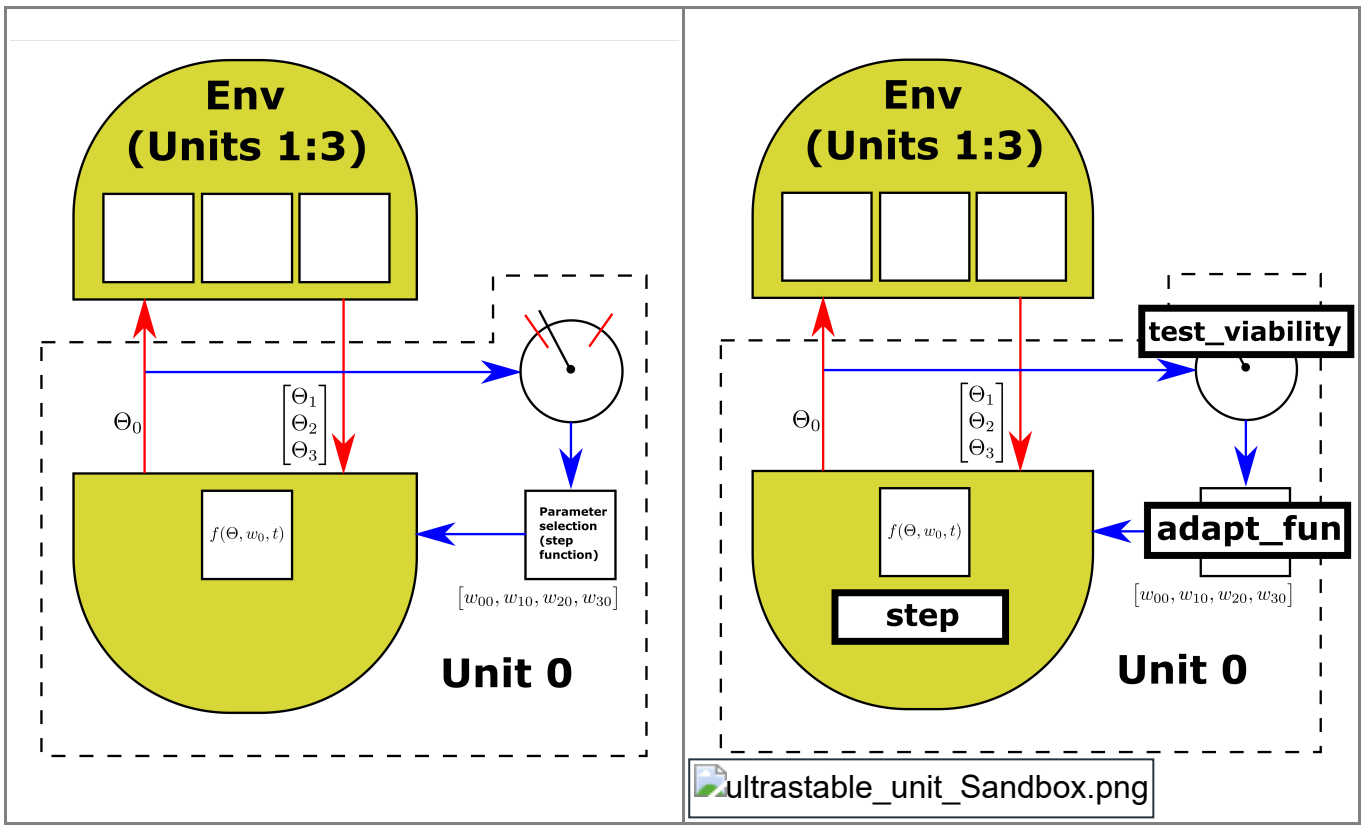

Figure 4. Each unit in the Homeostat is ultrastable, and - as illustrated on the left-hand side of this figure - from the point of view of any individual unit, all of the other units form its environment (Env). In the diagram on the right-hand side we see how some of the functions of the `Homeostat` class map to the ultrastable system.

Each unit in Ashby's Homeostat had a discrete set of connection weights which it could randomly select from when it was adapting. You don't need to understand Ashby's electrical circuit, but if you're interested, the weights were physically realised using electrical resistors, which only come in certain sizes, and this is why Ashby was constrained to use a discrete set of predetermined values. We don't have this limitation when simulating a Homeostat on a PC, as we have pseudo-random number generators at our disposal, which can produce what appear to be (to a high degree of precision) random numbers drawn from a continuous interval.

In the previous lab, you were introduced to the `step_fun` function of the `Controller` class. This function works as a kind of "plug-in" for a controller, which can be used to determine its usual mode of operation, in the first order feedback sensorimotor loop. The `Controller` class also has a plug-in called `adapt_fun` which can be used to make a controller *self-adaptive*. For consistency, I also gave the Homeostat class an `adapt_fun` function, which can be used to implement and experiment with different self-adaptive processes for a Homeostat unit.

I have implemented 3 `adapt_fun` functions for you to study and compare. You can find their implementations at the bottom of the `Homeostat.py` file. You can select which function to use by setting the value of `adapt_fun` in `params.yaml` (**Figure 3**), They are:

1. `random_val,` which sets all of the weights to random numbers, drawn from the interval [-1, 1].

2. `random_creeper,` which adjusts all of the weights by a small random amount, drawn from the interval [-0.05, 0.05]. Not that the effect of this is drift - just like the brown noise which we have seen in other labs.
3. `random_selector,` which works similarly to Ashby's uniselector, as it randomly selects weights from a set of discrete values which is passed to a `Homeostat` when it is constructed.

Tasks

1. Compare the different `adapt_fun` functions. You can select which `adapt_fun` to use on line 7 of the params file (**Figure 3**), by changing the function name written there.
   - Which do you think works best?
2. With `random_selector`, try using larger or smaller sets of weight values.
   - Is a larger or a smaller set of values better?
   - Do you ever notice anything strange if zero is in the set of values?
   - Does a smaller or a larger range of values seem to be better?

Negative feedback loops

In every `adapt_fun` function which I implemented, you will see the same two lines of code just before the weights are returned. Those lines make sure that the feedback connection, where the output from a unit is fed back to one of the unit's inputs, is always negative. This weight doesn't have to be negative, but - as we know - negative feedback is associated with stability, and making these weights always negative leads to the Homeostat finding stability faster than when they are either positive or negative.

Tasks

1. Try commenting those lines out and seeing what difference that makes. [Note: this is the one part of this lab where you have to edit the program code in `Homeostat.py`, rather than just modifying `params.yaml`].

Boundaries and limits

The limits, or boundaries, of the viable region are specified by `lower_viability` and `upper_viability` (**Figure 3**). There are also hard limits to how far the essential variable can move, and these are specified by `lower_limit` and `upper_limit.`

Tasks

1. Investigate the effects of changing these parameters. Questions that you may consider include:
   - Does the region of viability need to be centred on 0 in the Homeostat?
   - What difference does it make if either or both of the hard limits are inside the region of viability?
   - What difference does it make if either or both of the hard limits are *much* larger than the limits of viability?

Time to wait for stability

Each time Ashby's Homeostat's weights were changed, there would be a period of time during which the weights would not change again. We can think of this as giving the system time to return to viability - changing the weights changes the lines of behaviour, but even when a stable field is found, it may take some short period of time for those lines of behaviour to return the system to the viable region. The period of time which our simulated Homeostat will wait for after changing its weights is specified by `test_interval` (**Figure 3**).

Tasks

1. What difference does it make when you increase the `test_interval` parameter?
2. What difference does it make when you decrease that parameter?
3. What happens if you set that parameter to zero?
    - How do you explain what you observe?

The padlet for this lab is here:

Trouble viewing this page? Go to our diagnostics page to see what's wrong.
Skip to content

Lab 3 padlet

CHRIS JOHNSON 2YR

if you have any questions about the lab, or how to run the code, please ask them here

0This post has 0 upvotes
0This post has 0 downvotes
0This post has 0 comments
Add comment

Do you have any nice plots that you would like to share? For example, ones that show interesting behaviours, or just make nice patterns

0This post has 0 upvotes
0This post has 0 downvotes
0This post has 0 comments
Add comment

Made with Padlet