Introduction to Complexity and Applied Complexity, Spring 2021

Module 2 — Introduction to Dynamics

Notes by Sav Sidorov

Readings

- An Introduction to Cybernetics, Chapters 1 to 3 (with some ideas from chapters 4 & 5) — W. Ross Ashby

On Cybernetics

The readings for this module are the first three chapters of W. Ross Ashby's *An Introduction to Cybernetics*.

Direct quote from Joe on Ross Ashby:

> "This is an extremely useful book if you are interested in getting your head around the mathematics involved in systems science. Ross Ashby has a gift for taking things that — when presented elsewhere — seem impenetrable and making them seem like no big deal. I highly respect that gift. It turns out that very few have it, and it's helped me work through some of this stuff. This is actually a book that I return to probably once a year or so — not necessarily to work through front to back, but specific sections. Whenever I've run into something that I don't quite remember, or where I don't quite get what they're doing here, there's almost always a chapter or section in *Cybernetics* that addresses it in a very clean way. Ross Ashby is a name that not many people are familiar with, but he's a giant in cybernetics.

> Cybernetics was kind of subsumed by complex systems science, serving as the foundation to be built upon. I like a lot of the spirit — the geist — of the cyberneticians. They're kind of out there, it seems like they were taking acid or something. They're fun. And of course, out of this tradition comes Shannon's information theory, which is foundational to the world we live in today."

The quote pulled from *An Introduction to Cybernetics* is:

> *"[Cybernetics] does not ask what is a thing, but what does it do?"*

That's the fundamental framing of cybernetics. As we've been talking about, we're not looking at *things* — we're interested in the patterns, the interactions, the behavior.

The Twin Pillars of Newtonianism

There are two very important kinds of expectations — informal assumptions — that people have. Both are associated with what we'll call Newtonianism. What we'll do is, very explicitly, break down these expectations using nothing but the tools of Newtonianism itself.

Newtonianism is this idea that there's a fundamental set of laws, and these laws dictate all of the changes that we see. Each change is a necessary consequence of the state of the system now; you have the state of the system, you have a law, and based on the law, you get to the next state of the system. That's the essence of the world according to Newtonianism.

The two pillars of Newtonianism are:

- **Laplace's demon.** The idea being, if I know the governing laws, and if I know the state of the universe, then I know two things: I know everything that's going to happen, and I know everything that has happened, because each is a consequence of the laws and the state.

This expectation breaks down, at least for practical reasons. We will see a parting of ways between determinism and predictability.

- **When we look at the behavior of any part of the universe, it's completely accounted for by external forces applied to that piece of the system.**

  But as it turns out, complex systems *generate their own behavior*, their own forces even. If this is the case, then the behavior of a system cannot be accounted for by only looking at the external forces applied to it. Complex systems behave in this way due to their internal organization.

Along the way we will gain other useful concepts related to the idea of change more generally: attractor dynamics, self-organization (like the water/ice example from Module 1), stability and instability, and a foundational set of formalisms that underlie much — if not most — of physical theory.

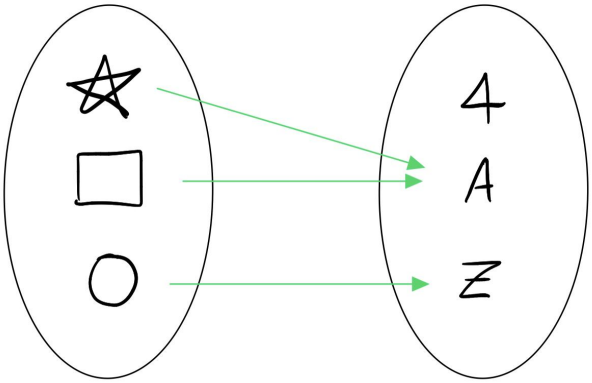
In order to understand how people model these dynamics in the Newtonian way, and how these models break down, we will need to expand our mathematical toolkit. Let's first start with functions.

Functions

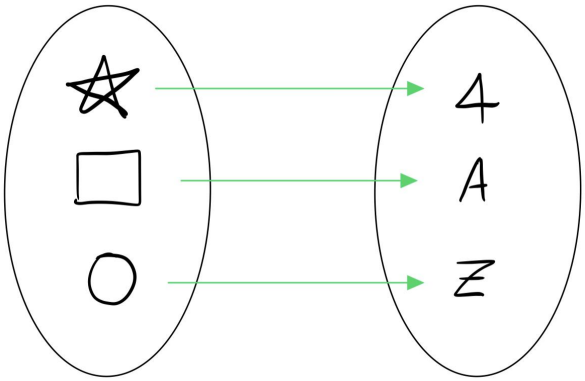
What is a **function**, mathematically speaking? It's easiest to enter this idea through the idea of **sets**.

So then, what is a set? A set is a collection of things. Often when you're doing mathematics, they end up being numbers. But let's make them shapes in this case — a *star*, a *square* and a *circle*. And in another set we'll have numbers and letters — *4*, *A*, *Z*. All a function is then, is a mapping, where there's an arrow going from an item in one set to an item of the other set. The only constraint we have here is that on the left, for each of the items in a set, there can only be one arrow coming out. If there's more than

one, it's not a function. But it's not necessarily the case on the other end. We can have multiple arrows converging and still have a function.



One thing to note is that if I wanted to have an inverse of my function — that is, to reverse it — with a situation like we have now, I can't actually do that. That's because if we want to go the other direction, we don't know whether *A* maps to the *star* or the *square*. However, if we have a one-to-one function, instead of a many-to-one like in the first drawing, we can reverse it. Now we can have both a function going from left to right, and from right to left.



And of course, we could have a mapping that's one-to-many, but that wouldn't be considered a function.

Discrete vs Continuous

One other thing to note about each of our sets here is that they're **discrete sets**. This means that they're countable — there's no ambiguity around what's in this set and where the objects are. However, sets can be continuous, like the real number line. On the number line, there's an infinity of values between zero and one. No matter how far you go down, there's always something in between.

The difference between **continuous** and **discrete** is an important distinction in our formalisms. As we go along, we'll make clear whether we're talking about something continuous or discrete — it could be time or space, for example. Sometimes, you'll have a mixture of the two in the same model — say, continuous space and discrete time. It's good to be aware of that difference.

It's typically much more digestible to approach things and learn about things through a discrete representation, so that's what we'll rely on wherever possible.

Functions and Formulas

One of the things to draw a point to here is that, when people usually think about a mathematical function, they're not thinking about what we just drew — this mapping of arbitrary objects. In our example, there's no clear rule as to *why*, for example, *circle* maps to *Z*. Typically, when someone says mathematical function, people think of something like this, where there's some formula that produces the mapping:

$$y = f(x) = x^2$$

What this notation means is that there's a function *f*, and it has an argument *x*. It's like a little machine, almost. You put the *x* into the machine and it gets mapped to *f(x)*. In this case, the machine outputs the square of the value. A formula like this is just a compression of a function — it's a compact way of denoting a mapping over sets.

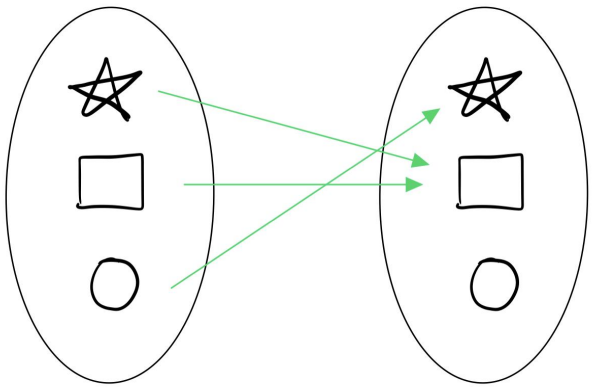
Formulas are especially useful in the continuous case. How are you going to write down every mapping if there are infinitely many things in your set? You need some way of generating that mapping by some rule — some formula.

Here again we see the inversion of what's considered special and what's considered general. For instance, when we talk about functions as mappings between items and sets, this is not a special case, but actually a very general case. Through this lens, formulas — as a way of encoding functions — become special cases. Not every function can be encoded as a formula. The same notion comes up when we talk about discrete and continuous representations. Typically, we think of continuous spaces as the general case, and discrete as more special. But in fact, it is discrete math that can always approximate continuous math. You might need to lower the resolution to get to that appropriate approximation of something continuous, but it does work. It doesn't work as well the other way around — it's very difficult to approximate a discrete system with a continuous system.

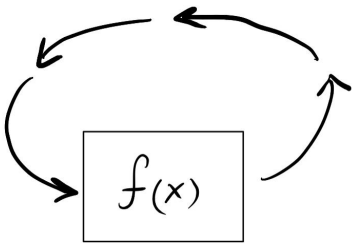
Closure and Recursion

Getting back to our two sets, in some cases, the transformation between sets maps from some set *back to that same set*. This property is called **closure**.

Another way to think about closure is that all of the elements of the output set are found in the input set — all outputs can serve as inputs.



If a system has closure, we can loop around indefinitely and not worry about getting stuck at some output.

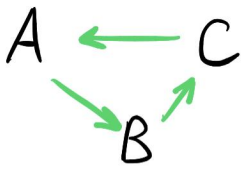


This property of closure is something that will come up in a lot of different contexts. It doesn't always imply closure *on a function* like we're talking about here. Closure turns out to be really crucial to what we talked about with the Newtonian take on material reality.

When we talk about physical laws — the laws of the universe — we're really just talking about mappings. When you put something in, you get something out. And the idea of closure is crucial to our picture of the universe. If you can't put whatever you get out back into the system, then the laws can't tell you what's going to happen next. If we represent reality as a kind of machine, then a lack of closure would indicate something like the machine breaking or jamming — something going wrong. Or, looked at another way, lack of closure has to do with a failure of the dynamical law to account for the behavior that we observe.

Kinematic Graphs

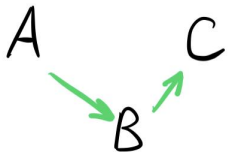
As we mentioned before, a system can have different states. We'll label them with letters. We also have transitions between states that are manifest by the law (the function). Assuming that the transformation is closed, we don't need to draw two different set representations with arrows between them — we can just draw the arrows directly between the states of the system. We call this representation a kinematic graph. A few examples:



These graphs represent the dynamics of the system.

If we want to simulate the dynamics of the system, we would need to indicate an initial condition. For example, let's start the system on the top left in state *B* and see what happens. If we start at *B*, we'll go to *C*, then go to *A, B, C*… cycling indefinitely.

Now let's assume we don't have closure. What that would mean, in this case, is that there's no arrow coming out of *C*. *C* can't serve as an input to some other state — you can't start at *C* and go elsewhere. Our system has nothing to say about what happens after we reach *C*.



So then, if we want our system to represent continuous unfolding in time, it needs to have closure. To give this system closure, we might draw an arrow from C back to itself. Often, however, when we draw kinematic graphs, we leave out these self looping arrows. *If a state doesn't have an arrow coming out of it in a kinematic graph, we're implying that there's a self loop here.*

One of the things that becomes more apparent over time is that these graphs are not just mappings. The character of the mapping is manifest in the organization of the diagrams. In other words, we can very quickly see what behavior of the system emerges from the graph. The mappings are not just mappings, they also have implicit organization due to closure. For instance, in the bottom graph of our three examples, if we started the system in *D*, it's going to transition to *C*, transition to *B*, and then it's going to oscillate between *A* and *B* forever.

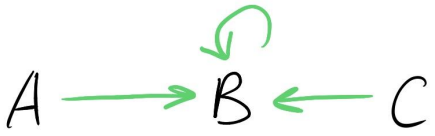
Here it's made clear that the behavior of these systems is completely one-to-one with the organization of the system. The organization *is* the behavior, the behavior *is* the organization.

We can also say that the system is generating its own behavior — the behavior is not happening due to some external force.

One of the big takeaways from a lot of this material is that many of the deepest insights come from simply taking ideas that people have been working with for a long time in one way and finding better representations for them. Better representations make the properties we're interested in become more obvious.

Reversibility and Irreversibility

Let's say we have this *A*-*B*-*C* system:

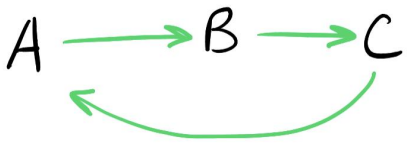


Wherever you start, on the next step you're going to be in state *B*. In a sense, this is not a reversible system — once you're in *B*, you can't go to other states. At *B*, you also don't know what state the system was in before *B*. So **reversibility**, in a sense, is the idea that

one can trace their steps and figure out exactly where they came from. It turns out that this is an extremely generic property of systems — systems can converge to a state from multiple other states.

> As an aside: This says something really profound about our ability to track history. How do we think about the events that led up to us being here? Primarily, we depend on written and oral accounts. But if you look at the state of things today and try to figure out where everything came from, it might be impossible to do so. It might theoretically be possible to trace every particle, but that doesn't matter for practical purposes.
>
> We often ask the question: "How hard is it to predict the future?" It turns out that *it's equally hard to postdict the past*, even in a deterministic world. In fact, because of this indeterminacy of paths going backward, if we lived in a deterministic world, it would actually be easier to predict the future than to talk with certainty about what happened in the past.

The *A*-*B*-*C* system we introduced is **irreversible** — you can't run it backwards in a deterministic fashion. Here, on the other hand, is an example of a **reversible** system:



One way to look at it is to see how the system branches. If it branches going backwards in time (e.g. if there are two or more arrows leading into *B*), it's irreversible.

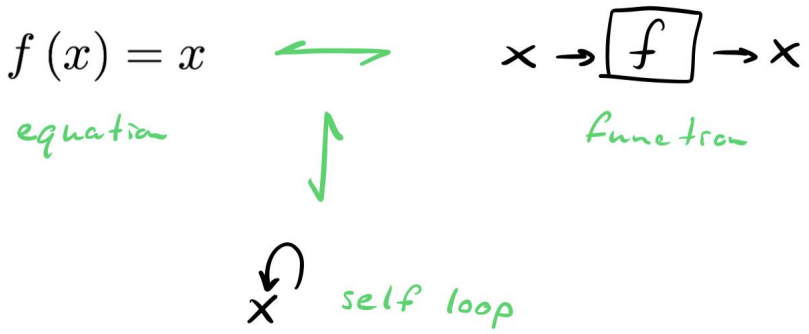
Keep in mind that determinism is not enough for reversibility. Everything we're drawing here is deterministic, but it's not necessarily reversible.

Fixed Points

One of the things that's been implicit in a lot of the examples we looked at is the idea of fixed points. A **fixed point** is when your mapping satisfies the following:

When you put x through a function, you get x out.

This idea can be represented by an equation, a picture of a function as a machine, or a self-loop in a kinematic graph:



Another name for a fixed point is **equilibrium point**.

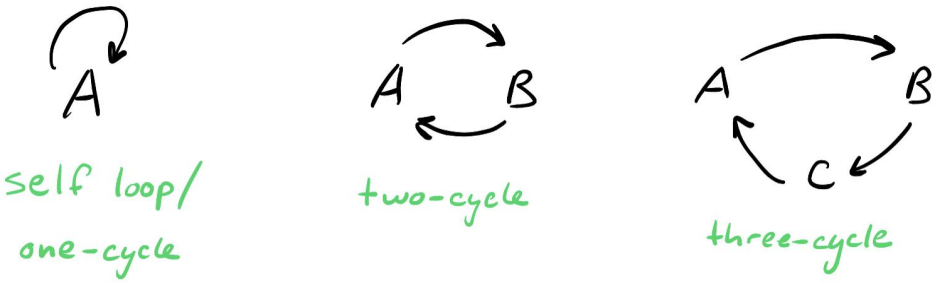
We often look for fixed points when doing mathematics. One of the ways to find a fixed point is to take a system and recurse on it until you reach one. For example, you could go on a walk in our irreversible *A*-*B*-*C* system until you reach *B*, where recursing further would tell you that you only get *B* back with each successive step. You can't climb out of a fixed point.

A fixed point is a state of the system that maps right back to itself.

Cycles

We also mentioned this idea of cycles. When we talk about a cycle, all we're talking about is the notion of things returning to some previous state. A fixed point is actually

a special case of a cycle — a sort of perfect self-mapping. But there are other types of cycles out there as well. We can, for example, have a two-cycle and a three-cycle:

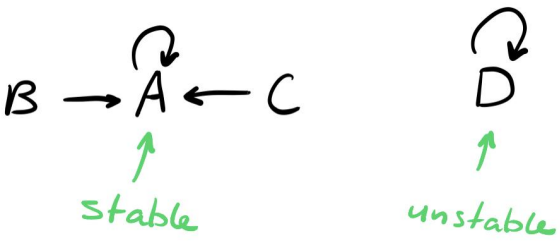


Stable vs Unstable Fixed Points

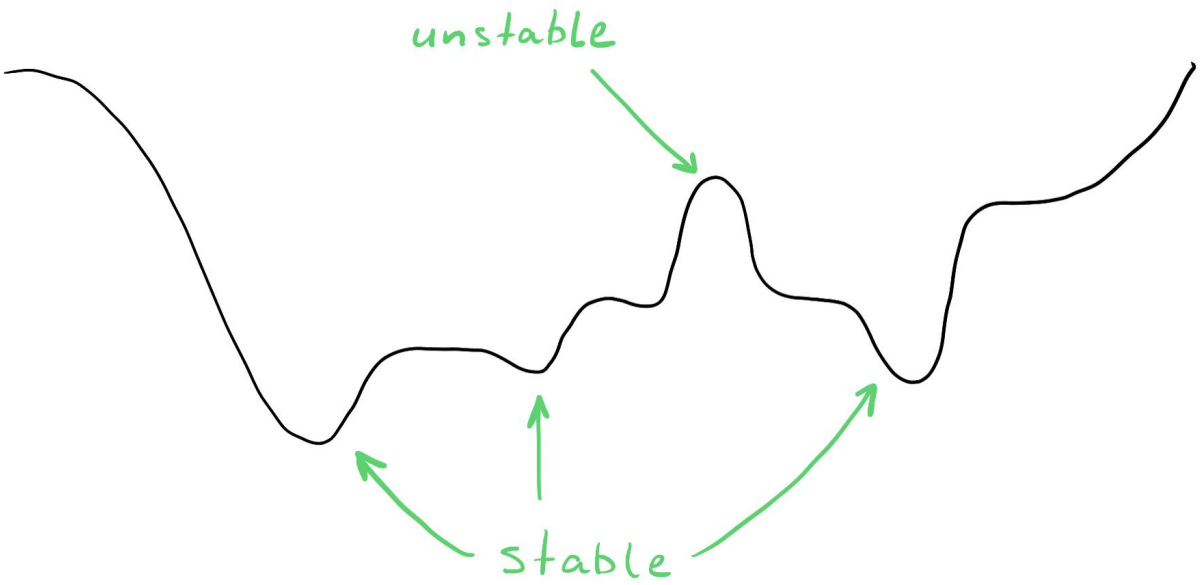
Let's talk about this idea of stability and instability. Imagine we have a system with a fixed point at *A*:

No matter what state we start in, we return to *A*. This is the notion of **stability**. For stability, it's not enough to have a fixed point in the system. When you push the system out of that fixed point, it has to return to it.

Now imagine we have some other state *D*. State *D* is another fixed point. If the system is in *D*, it will stay in *D*. But if pushed to any of these other states, the system will not come back to *D*. We can call this an **unstable** fixed point.



We can represent fixed points on a landscape. A ball balancing on this landscape is in equilibrium at the top of a hill, or at the bottom of a valley. When pushed, however, it only returns back to the valley, not the hill. Valleys are stable, hills are unstable.



This will come up again, especially as we go into Nassim Taleb's work around fragility and antifragility.

> Aside: there's a huge conflation that happens related to this idea — especially when you look at political science and political commentary. It goes something like this: *because something hasn't moved in a while, that implies that it's stable*. But of course, as we're seeing here, that's not necessarily the case at all. You could be one little nudge away from never returning to that state again.

Let's look at an example of stability in nature: think of branches blowing in the breeze. Branches are very much willing to bend with the breeze. Of course, there's a limit to that — if the wind blows hard enough, it can snap them. But they're much more stable than rigid things. Having a wind push against something rigid with enough force can

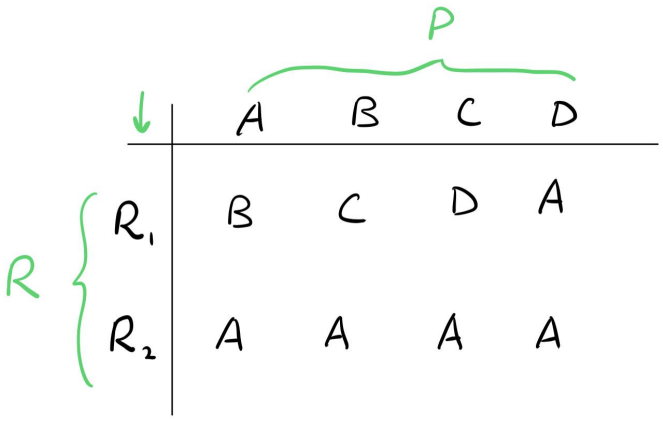
break it, because rigid things can't bend. Something rigid isn't necessarily qualitatively different from something flexible. It's just that the valley of stability is much narrower. You can push it, but it breaks much quicker.

We can also refer to stable fixed points as **basins of attraction** — regions over which the system will return to its previous state.

The System with Input

So far, we've only been talking about the system by itself — what kind of behavior does the system exhibit on its own? But of course, systems can have inputs. What we want to do is not identify the system in total, with a given mapping, but rather suggest that the system is a set of mappings that depend on the input.

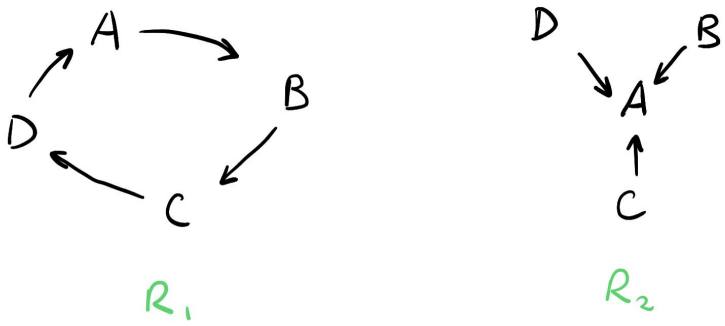
Imagine we have a system that can take on four states: *A, B, C, D*. We'll call this system *P*. If the system is in *A*, it goes to *B*, if in *B*, goes to *C*, and so on, in a loop. Call this set of transitions R_1. However, we can have a totally different set of transitions, called R_2 for example.



We can call these different sets of transitions — R_1 and R_2 — by two different names: the **inputs** or the **parameters** of a system. When we're thinking in terms of cybernetics, these two things are essentially the same.

So the system, in our way of looking at it, is the set of states plus the input. For example *A, B, C, D + R_1* or *A, B, C, D + R_2*.

This kinematic graphs for R_1 and R_2 look like this:



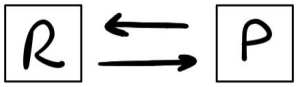
So now we're associating the mapping not with this total description of the system, but with the input. This idea is important, because these inputs could, of course, be states of some other system.

Reducibility

This leads us into **coupled systems** — one system can be behaving under its own laws, while serving as the inputs for another system. The notion of whether a system is coupled or not gives us a good way to talk about reducibility.

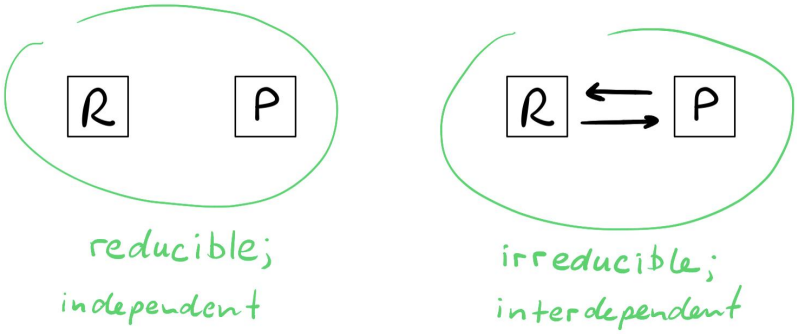
When talking about systems P and R, we can indicate that R serves as input for P by drawing an arrow from the box representing R to the box representing P:

Often, in complex systems, you'd get bidirectional coupling, where each system is influencing the other:



Reducibility then, in this picture, is associated with there being no arrows between systems *P* and *R*. In a world like that, we could reduce the system to just be *P* and look at *P* independently. We could do the same for *R*.

In a world where the two systems are **irreducible**, however, you couldn't just look at *P* or *R* in isolation — the systems depend on each other.



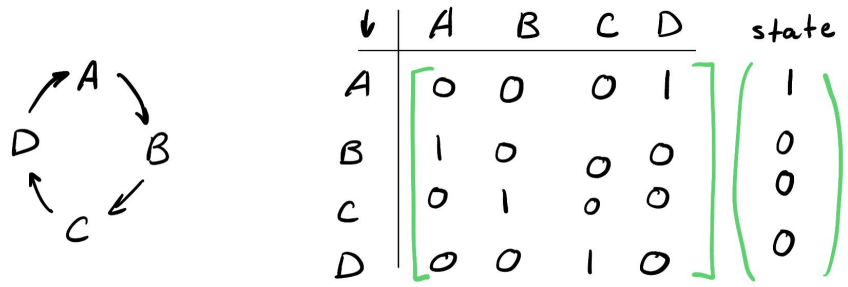
We'll see this idea of independence & interdependence come up again and again. It's very central. To paraphrase Ashby, thank goodness that the world generally is not purely irreducible — that there exist independent systems that we can study and understand. If the entire world were irreducible, we wouldn't be able to navigate it.

> Aside: what's interesting about this sentiment is that now, in 2021, we practically *do* have a globe whose behavior is irreducible. It used to be reducible to a much greater extent, but because everything is much more interconnected now (both physically and informationally), that is no longer the case.

Matrix Representation

Let's go through one other way of representing these transformations. This will not add anything useful immediately, but it will set us up to generalize to some things later.

We can represent the system as a matrix. Take, for example, the system *A-B-C-D*.



When the state [1, 0, 0, 0] — corresponding to state *A* — is fed through the system, we multiply the vector through the matrix and get [0, 1, 0, 0] — state *B*.

Difference Equations

Difference equations are the discrete-time equivalents to differential equations. Each current state of the mapping is a consequence of only the previous step. For example:

$$x[t+1] = f(x[t])$$

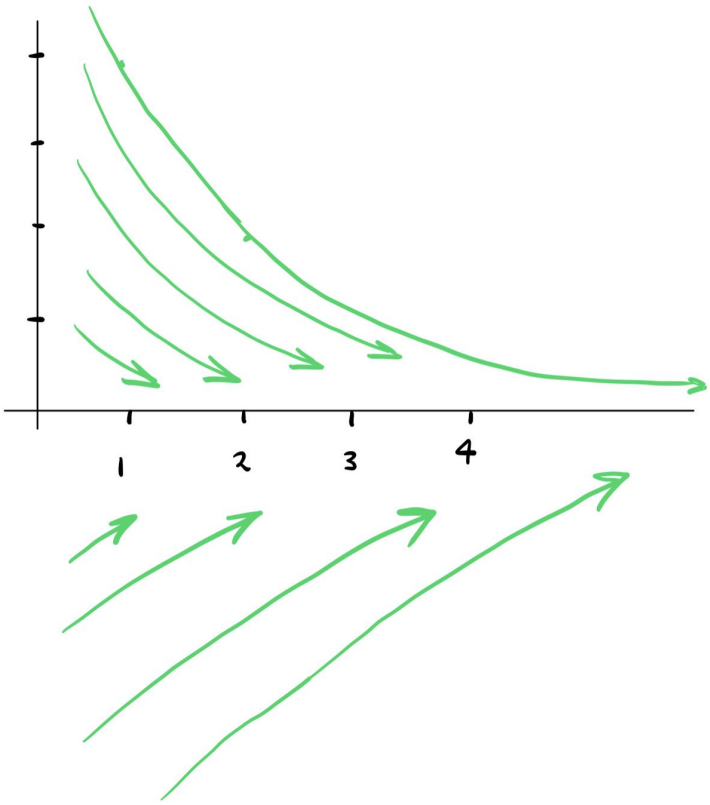
$$f(x) = x + 1$$

$$x[t] = 1$$

This might also be a good time to introduce the **Markov property**, which states that where you go next is only a function of where you currently are.

Here's an example of what we call a fixed point attractor, represented as a difference equation:

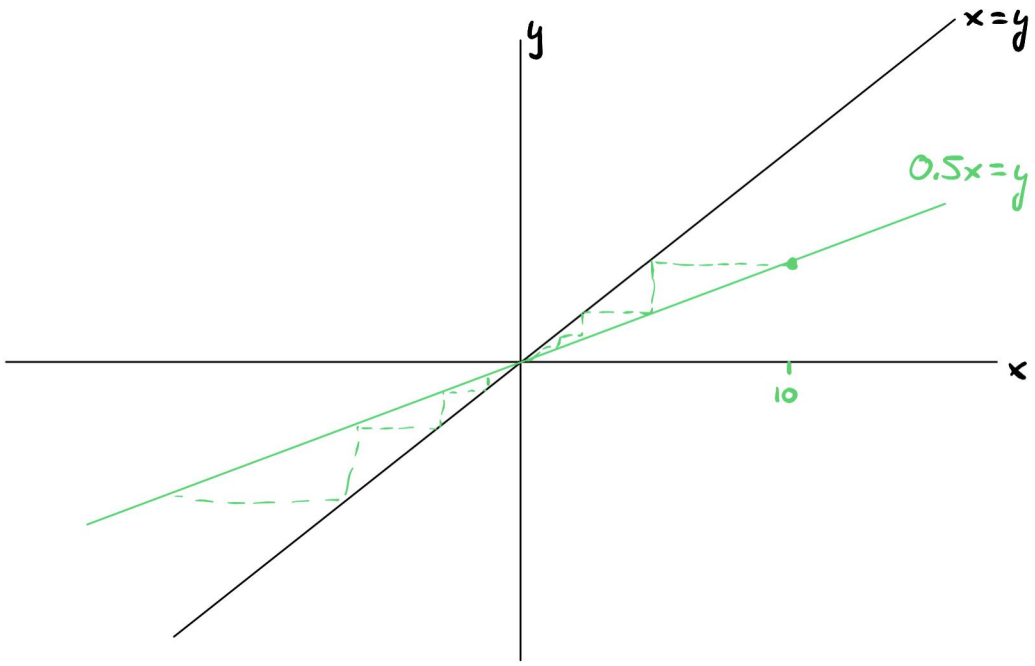
$$f(x) = 0.5x$$
$$x[t+1] = 0.5x[t]$$



Whatever value you pass in, the system halves it on the next time step (here, the x-axis represents time). If recursed upon, all values eventually tend to zero.

Phase and Cobweb Plots

A **phase plot** explicitly maps the current state to the next state — from the x- to the y-axis. The line $x = y$ is called the identity line. For the difference equation we're playing around with, we would have the following phase plot:



When little dashed lines are drawn to indicate the mapping. With this mapping included, we get what's known as a cobweb plot. Again, if you recurse upon this function by following the dashed lines, all values eventually tend to zero.