1D Discrete (Pt. II)

This week in NB4050

We deepen our study of 1D maps by understanding the emergence of higher period orbits. We study the stability of periodic orbits and inspect the phenomenon of period doubling cascade to chaos. We analyze the sensitivity of orbits to initial conditions for complex orbits and define a mesure for chaoticity, the Lyapunov exponent.

Learning Goals

- Analyze the long term behavior of a parameterized dynamical system.
- Explain the emergence of period doubling in 1D maps.
- Understand the relationship between periodic attractors and iterated maps and analyze stability of periodic orbits.
- Enumerate the hallmark properties of a chaotic attractor.
- State the fixed point theorem.
- Calculate an approximation to the Lyapunov exponent.
- Grasp the concept of topological equivalence.

Stability of Fixed Points

(Ref. [AY], Theorem 1.5, [AY] Sect. 1.3 , [HK] Sect. 3.4)

Hyperbolic fixed points come in flavors, the most common of which are stable (which attract orbits), and unstable (which repel). The stability is given by the modulus of the slope f' of the map f at the fixed point x^*. That is,

- stable: $|f'(x^*)| < 1$
- unstable: $|f'(x^*)| > 1$

The proof of this fact is simple. Imagine that we have a slope of the derivative (call it 'a') at the fixed point ('p') whose modulus is smaller than 1. Now, taking the definition of a derivative and simply stating that the derivative is smaller than a:

$$lim_{x\to p}\frac{|f(x)-f(p)|}{|x-p|} = |f'(p)| < a$$

It should be evident that for any x very close to p (say, ϵ):

$$\frac{|f(x)-f(p)|}{|x-p|} < a$$

meaning that x is closer to p than $f(x)$ is to $f(p)$:

And the more we iterate the function (k times), the closer they become, exponentially:

$$|f^k(x) - f^k(p)| < a^k|x-p|$$

Things look similar to the proof for unstable fixed point (a>1).

Attracting and repelling fixed points with $|f'(x^*)| \neq 1$ are called hyperbolic (because states close to the fixed point separate or approach according to a hyperbola).

And how about fixed points which are neither attracting nor repelling? When the modulus of the derivative at the point is exactly one? There, we see a...

Bifurcation

A **bifurcation** is a **qualitative change in the behavior** of the orbits of a map (or continuous function as we will see later) as a parameter changes. Bifurcations occurs when (1) fixed points change stability (from attracting to repelling, for example), or when (2) further equilibria emerge (for 1D maps this happens, for example, when a deformation of the function defining the map leads the function to cross the identity line).

It is often the case that when that when new equilibria appear, we observe the emergence of ...

Periodic orbits

([AY], Sec. 1.4, 1.5, Definition 1.8)

Periodic orbits often emerge when fixed-points appear in the higher order compound maps f^k, that is, when $|f^k(x^*)| = x^*$ for $k > 1$.

Stability of periodic orbits

Incidentally, the **stability of periodic orbits** can also be tested with the method of calculating the derivative of the compound map, $f^{k\prime}(x) = x$ — where k is the number of iterations of the map. The same rule for stability of fixed points applies to higher order maps.

To note:

- Fixed points of the iterated map may, or may not, be part of periodic orbits.

Bifurcation Diagram

(refs. [AY] , [H.K] 3.3 Bifurcations of monotone maps (Cases I, II and III) 3.3HK)

To discover the pattern of orbits under deformations of the logistic map by a parameter, we introduce the **bifurcation diagram**, which is a method of displaying the long term behavior of orbits for a range of parameters. Essentially, we take a parameter for a range and compute the orbit for a number of steps. In the logistic map for example, an interesting range for the parameter **a** would be [0,4].

This is the recipe to construct a bifurcation diargram of a given parameterized function:

1. create a range of possible parameters, the so called **parameter space** (one possible parameter space for the logistic map could be, for example 0..4)
2. computing an orbit for a certain number of iterations (N)
3. pruning the **transient** of the orbit (ditching the first K iterations)
4. plotting the remaining orbit as a function of the parameter

Coding Exercise: Bifurcation diagram with arrayfun

```
% Compute and display the bifucation diagram of the logistic map by using an
extended version of your function 'orbit'.
% You can extend the function 'orbit' by using 'arrayfun' to simultaneously
compute orbits for a range of parameters. 'arrayfun' applies a function to
every element of an array or multiple arrays.

% Requirements:
% - Use arrayfun with your function orbit to apply a function to every element
of an array such as [0:.1:4].
```

If the above is clear to you, you can skip the digressions on `cells` and `arrayfun` , but don't skip the fun of applying functions to arrays/ .

Coding Digression

Cell arrays

Most of the optimizations present in MATLAB are based on the assumption that the data provided is in the form of a matrix and is therefore square or rectangular in shape. However, this is sometimes not sufficient. For this purpose, the clever folks at MathWorks introduced what is known as the Cell Array, often confusingly referred to as a `cell`. The `cell` is a **linear**, **random-access** data collection able to contain **arbitrary types**.

- Linear: Items are stored in order, such that it is sensible to talk about a first, second, third etc element. The cell array has a length.
- Random-access: You can access any element at any time by indexing. Unlike the way you index an array in MATLAB by using the round brackets `( )`, you use curly braces `{ }` for `cell`s.
- Arbitrary types: An array can only contain the same data types, such as *only* characters or *only* numbers. A cell array does not have that restriction. It is possible to create a `cell` that has a number as its first element, a character as its second element and even a vector or matrix as its third. Two vectors in a `cell` don't even have to have the same shape or dimensionality. In order to declare an empty cell array, you can state

```
my_cell = cell(3, 1); % This makes a cell array of shape 3x1
```

To add an element, you index via

```
my_cell{1} = 1;
my_cell{2} = 'a';
my_cell{3} = [2 3 4];
```

and to get an element you can say

```
my_integer = my_cell{1};
```

Disclaimer:

The cell array is often, if not always, significantly slower than natively using vectors. The reason for this is that the MATLAB interpreter has to do many different **checks** on the data. If for example you want to take the sum of all elements in a cell array, the interpreter has to check if the `+` operator is even defined for all the types in the cell. In order to optimize for speed, it is better to use arrays. If we instruct you to use a cell array, be assured that there is no faster way, or, if there is a faster way, we find that this way obfuscates the main idea of the exercise.

Coding Digression: Functional programming and arrayfun

The way you have learned to program is known as **imperative programming**. This means that your code consists of **data**, as well as different ways to tell the interpreter what to do with your data, namely **functions**. This is however not the only way to write code. In your course Electronics and Instrumentation, and perhaps in other moments in your life, you have been exposed to **object-oriented programming**. This is a paradigm, started by Smalltalk in the 1980s, which decides to couple together data and functions into a single **object**. An object could for example represent a person, which has some data (age, name, gender etc), and some functions it could apply on its own data (changeName, ageUp). This paradigm therefore sees data and functions as linked. A completely different paradigm, which has been present in academia since the 1950s, startng with LISP, is **functional programming**. After a period of long obscurity, functional languages such as Scala and Wolfram's Mathematica are starting to become popular, and functional elements are added to languages such as Python, JavaScript and even MATLAB. A central tenet of functional programming is that the distiction between functions and data is fuzzy. This entails that data and functions can be used in a similar way, *i.e.* functions are **first-class citizens**. This has been shown in MATLAB in week 1 in the form of function handles. A function handle behaves similar to normal data, being able to be stored and passed. Functions can therefore be also be passed into other functions. Functions that take other functions as arguments are called **higher-order functions**, and are often useful in coding to express simple concept concisely. An example of a higher-order function that is often used is called a `map` function, in MATLAB called `arrayfun`. This is a function that is shorthand for the following code

```
function out = arrayfun(input_function, array)
    out = array;
    for i=1:numel(array)
        out(i) = input_function(array(i))
    end
end
```

What this code does is apply a function `input_function` to all elements in a list, and returns the output. For example, using the `abs` function, we see that

```
arrayfun(@abs, [-1 2 -3]) -> [1 2 3]
```

The function is passed as a function handle. The function can be a

- Built-in, such as `abs` or `cos`
- Named function, defined in a separate file
- Anonymous function, such as `f = @(x) x^2 + x-2`

For the default `arrayfun`, the function has to fulfil a certain criterion. It has to have what's known as **uniform output**. For example, it should always output an integer, or a matrix that's exactly 3x3. This is done because in that case, MATLAB can construct a matrix from it. However, there are some cases where this just does not work. For example, take a function that can return either one or two outputs. In this case, instead of calling

```
arrayfun(@my_function, my_container)
```

you call

```
arrayfun(@my_function, my_container, 'UniformOutput', false)
```

and instead of an array, a cell array is returned, where the `i`th element is the function applied to the `i`th element of `my_container`. Here again the usefulness of the array list has been shown.

Warm-up: The quadratic equation and arrayfun

Let $f(x) = x^2 + 3x + c$. Write a function that takes c as input and returns the real roots (*i.e.* the values where $f(x) = 0, x \in \mathbb{R}$) of this equation. Use the quadratic formula. Subsequently, use `arrayfun` to evaluate this function for the range $[-5, 5]$ taking steps of $\frac{1}{4}$. Use MATLAB to find the value for which $f(x)$ has a single real root.

Bifurcation diagram and Lyapunov exponent

The main exercise for today involves drawing a bifurcation diagram for the logistic map. We will lead you through this problem in a number of steps.

1. Obtaining a fixed point set

The first function we will work on is called `orbit_after_transients`. This method will take a single parameter, called `a`, which is the parameter of the logistic map $x_{i+1} = ax(1 - x_i), 0 \leq x_i \leq 1$. Use the previously written function `orbit` for a fixed number of iterations, and take a random value as the initial value. Then, to get rid of the transients, remove the first N elements. Then return the transient-free orbit.

2. Obtaining the fixed points for a range of different a values

We will now work in the function `bifurcationdiagram.m`. Subsequently, you generate a range of values for a. Which range is useful? Then, use an arrayfun to apply the function `orbit_after_transients` to this range. Is `'UniformOutput', false` needed in this case? Which container type does the output have?

3. Plotting the bifurcation diagram

The easiest way to plot a bifurcation diagram is in the form of a scatter plot. In this scatter plot, plot all orbit points x for the same parameter a, that is, at coordinates (a, x). You can do this by looping. Keep in mind the way you index your container types!

For clarity's sake, make sure that the points are identically colored and not too large. Furthermore, don't be uncivilized and label your axes. Use a figure handle since you will need to add a second plot underneath it, as well as keeping the debugger happy.

Period Doubling Cascade to Chaos

In the logistic map, as the parameter a is increased, the higher iterates start producing more and more intersections with the identity line. For instance, when $a = 3$, two things happen: (1) the stable fixed point loses stability ($f'(x^*) > 1$), and (2) in the second iterate of the map, two other intersections appear, both with slopes smaller than 1, meaning these two equilibria of the second iterate constitute a stable (periodic) attractor. When a grows, these intersections of the higher iterates appear in pairs (can you see why?), leading to a **period doubling cascade**.

Chaos

(refs. [AY] Sect. 3.1 and 3.5, [HK])

For certain parameterizations, orbits are very complex, and clearly **aperiodic**. These orbits display **sensitivity to initial conditions**, that is, initial conditions that may start infinitesimally close to each other (in matlab, **eps**), soon diverge wildly. One measure of divergence of orbits is given by the ...

Lyapunov Exponent

[refs. [AY] Sect. 3.1 See also [AY] 6.1]

The **Lyapunov exponent** is a measure of sensitivity to initial conditions of orbits, which **calculates the exponential separation** of nearby orbits (ref: Alligood, 3.1). **Chaos** is observed when the Lyapunov exponent is positive.

It is defined as the average of the logarithm of the product of the derivatives of all states in an orbit starting at x_1.

$h(x_1) = lim_{n \to \infty} \frac{1}{n} ln[\prod_{i=n}^{n} |f'(x_n)|]$

Using the definition, calculate the Lyapunov exponent for values of the parameter 'a' between [0..4], by calculating the local divergence of the points of the orbit.

Estimating the Lyapunov Exponent:

4. Plotting the Lyapunov exponent

For our final exercise, we will work in the file titled `lyapunov.m`. Using the previously written function `orbit_after_transients`, generate an array (or cell array if you so fancy) that contains the Lyapunov coefficient for varying values of `a`. A similar technique as in section 3. is recommended. Plot this one underneath the bifurcation diagram, at the same scale.

5. Run the unit tests

Run the unit tests! Romano didn't spend his Sunday afternoon for you to ignore these :'(. They are great checks for the correctness of your code!

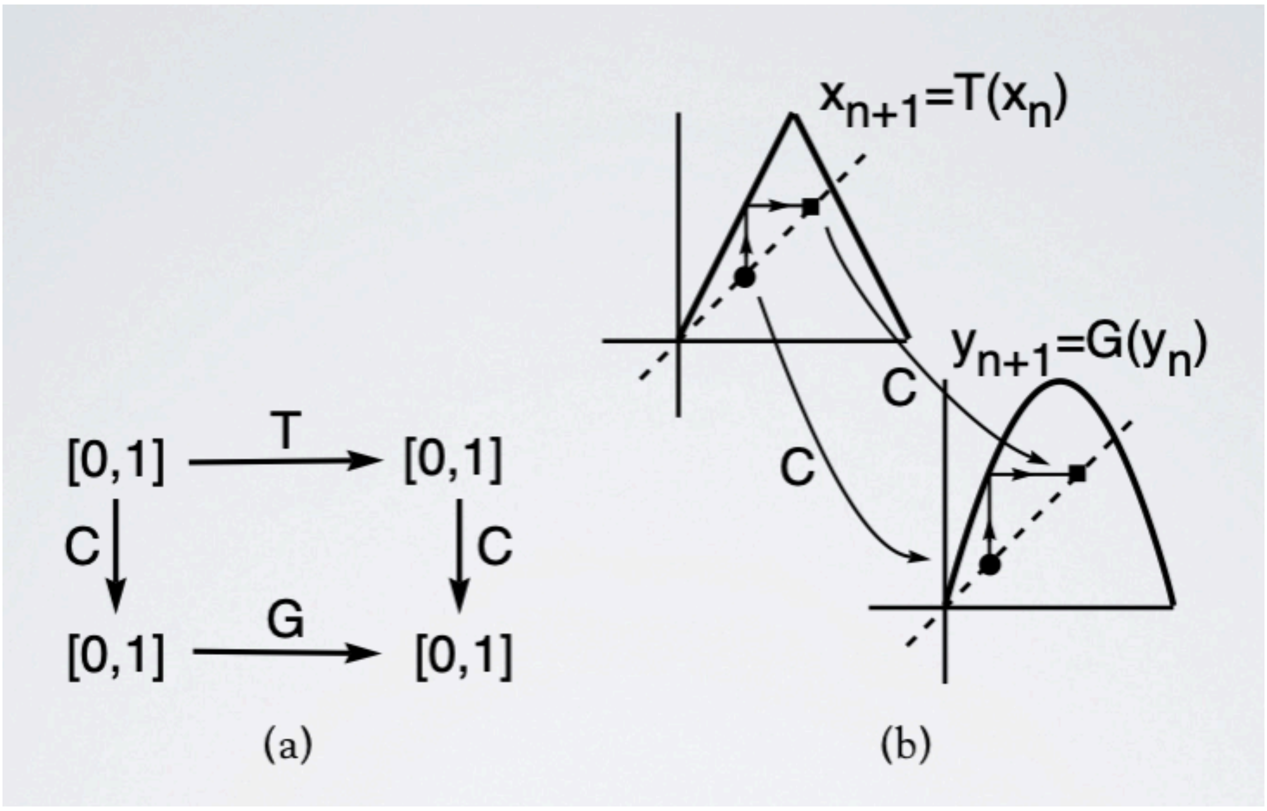
Questions:

- for which values of **a** is the logistic map chaotic?
- For a given dynamical system (a given **a** in the logistic map), is the Lyapunov exponent the same for orbits starting at any initial condition?
- After large values of **a** one observes positive Lyapunov exponents. But there seem to be intervals where the Lyapunov exponent 'dips' and sometimes become negative. What happens at those intervals?

Topological conjugacy

(refs. [AY] sec. 3.3 and definition 3.10, exercise 3.7)

When two maps F and G are topologically conjugate, that is, when there's a smooth transformation C that takes all the points in one map to points in the other map, the qualitative dynamics of one tells tales about the qualitative behavior of the other. This crucial fact is one of the cornerstones of the study of dynamical sytems, as it guarantees that many theorems have general applicability if functions are topologically conjugate.



References and Further Reading for This Week

[AY] Chaos - Aligood and Yorke (Chap. 1 and 3)

[HK] Dynamics and Bifurcations - Hale and Koçak

[1] The Birth of period 3, Saha and Strogatz.

[2] Periodic orbits (Chap.4 by Robert Gillmore) (Sect. 4.1 - 4.5, and 4.7.2)

[3] Sarkovski Theorem, [pdf] The Logistic Map, Period-Doubling To Chaos

Video Lecture

Steven Strogatz further elucidates:

- Geometrical and algebraic aspects of periodic orbits in the logistic map: https://youtu.be/pqJL_K2sS3Y
- Universality in 1D maps: https://youtu.be/ol6aQcgohxI