Homework 5

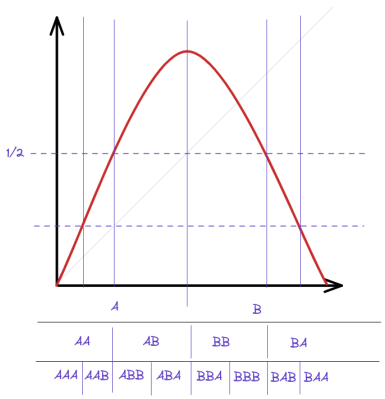
Information Theory for Complex Systems

March - 2023

a)

In order to show that $A = [0, 0.5], B = (0.5, 1]$ is a generative partition we need to show that $\text{diam}(x_1 x_2 \dots x_m) \to 0$ as $m \to 0$ as we consider sequences of increasing length.

As can be seen in the figure to the right, whenever we add another symbol to the sequence each interval is split into two subsets. As we can observe, there's always one subset with an A added at the end and one with a B at the end. The partition therefore enumerates all permutations of A and B uniquely. We get no disconnected sets, therefore all new splits shrink in size, equivalent with $\text{diam}(x_1 x_2 \dots x_m) \to 0$ as $m \to 0$.



b)

Starting at $x_0 = 0.2$, iterating the logistic map for 100 000 steps results in $P(A) = 0.51$ and $P(B) = 0.49$.

c)

I use the given ergodic measure and plug it into equation (8.6) from the course book which gives

$$\lambda = \int \mu(x) \ln |f'(x)| dx = \int_0^1 \frac{\ln |4((1-x)-x)|}{\pi \sqrt{x(1-x)}} dx = \ln 2, \tag{1}$$

which I calculated in Mathematica.

d)

Reading the time series and calculating the probabilities using the original A and B results in $P(A) = 0.36$ and $P(B) = 0.64$). This asymmetry doesn't match up at all with the simulated results. Switching the partition to $A' = [0.324, 0.6004]$ and $B' = (0.7884864, 0.9]$ results in $P(A') = 0.50$ and $P(B') = 0.50$.

Appendix: Source Code

All code is additionally availible at `https://github.com/vincent-uden/tif150-information-theory-for-complex-systems`

In[10]:=
```
r = 4;
f[x_] = r x (1 - x);
μ[x_] = 1 / (π √(x (1 - x)));
D[f[x], x]
```

Out[13]=
```
4 (1 - x) - 4 x
```

In[15]:=
```
Integrate[μ[x] * Log[Abs[D[f[x], x]]], {x, 0, 1}]
```

Out[15]=
```
Log[2]
```

main.rs

```
use std::{
    path::{Path, PathBuf},
    str::FromStr,
};

static R: f32 = 4.0;

#[derive(Debug, serde::Deserialize)]
struct Record {
    x: f32,
}

fn main() {
    let (p_a, p_b) = p_a_b(0.2, 100000);
    println!("P(A)={:.2}, P(B)={:.2} (simulated)", p_a, p_b);

    let p = PathBuf::from_str("./pop_series.csv").unwrap();
    let (p_a_csv, p_b_csv) = p_a_b_csv(&p, (0.0, 0.5), (0.5, 1.0));
    println!(
        "P(A)={:.2}, P(B)={:.2} (csv, A=[0     , 0.5    ], B=(0.5       , 1.0])",
        p_a_csv, p_b_csv
    );

    let (p_a_csv2, p_b_csv2) = p_a_b_csv(&p, (0.324, 0.6004), (0.7884864, 0.9));
    println!(
        "P(A)={:.2}, P(B)={:.2} (csv, A=[0.324, 0.6004], B=[0.7884864, 0.9])",
        p_a_csv2, p_b_csv2
    );
}

fn logistic_map(x: f32) -> f32 {
    return R * x * (1.0 - x);
}

fn p_a_b(x0: f32, iter_lim: i32) -> (f32, f32) {
    let mut a_count = 0;
    let mut b_count = 0;

    let mut x = x0;
    for _ in 0..iter_lim {
        if x <= 0.5 {
            a_count += 1;
        } else {
            b_count += 1;
        }

        x = logistic_map(x);
    }

    return (
        a_count as f32 / (a_count + b_count) as f32,
        b_count as f32 / (a_count + b_count) as f32,
    );
}

fn p_a_b_csv(path: &Path, a: (f32, f32), b: (f32, f32)) -> (f32, f32) {
    let mut rdr = csv::ReaderBuilder::new()
        .has_headers(false)
        .from_path(path)
        .unwrap();

    let mut a_count = 0;
    let mut b_count = 0;

    for record in rdr.deserialize::<Record>() {
        let x = record.unwrap().x;
```

```
        if a.0 <= x && a.1 >= x {
            a_count += 1;
        }

        if b.0 < x && b.1 >= x {
            b_count += 1;
        }
    }

    return (
        a_count as f32 / (a_count + b_count) as f32,
        b_count as f32 / (a_count + b_count) as f32,
    );
}
```