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,
    );
}
```