A small library for finite discrete-time Markov chains

Group N

Abstract

We present `markov`, a small Python library for finite discrete-time Markov chains. The library represents a chain by its transition matrix as a `sympy.Matrix` [2] and exposes three short functions: a stochasticity check, the distribution after n steps, and the stationary distribution. We illustrate the library on a two-state chain whose stationary distribution we compute exactly as $(4/7, 3/7)$, and we verify numerically that the distribution after 50 steps from a deterministic start agrees with the stationary distribution to within 10^{-10}. The library follows the Diataxis framework for documentation [4] and the matrices chapter of the open-source textbook *Python for Mathematics* [1].

1 Introduction

A discrete-time Markov chain on a finite state space is a stochastic process where the next state depends only on the current state and not on the history. The chain is described by its transition matrix P, a row-stochastic matrix where P_{ij} is the probability of moving from state i to state j in one step [3]. The distribution of the chain after n steps, starting from the row vector π_0 of initial probabilities, is

$$\pi_n = \pi_0 P^n.$$

A stationary distribution is a row vector π such that $\pi P = \pi$. For an irreducible, aperiodic chain on a finite state space, the stationary distribution exists and is unique, and the distribution after n steps converges to it from any starting point [3].

We wanted a small library that we could use to inspect the distributions of a chain step by step and to compute the stationary distribution exactly. The `numpy` package can multiply matrices quickly but loses the exact rational form of the answer. We chose to build on `sympy.Matrix` instead, following the conventions of the matrices chapter of *Python for Mathematics* [1].

We present a three-function library, `markov`; we recover the stationary distribution of a two-state chain in closed form; and we show numerically that P^n applied to a deterministic initial distribution agrees with the stationary distribution to within 10^{-10} after 50 steps.

2 The library

The library lives in the single module `markov.py` and exposes three functions.

`markov.is_stochastic(transition_matrix)` returns `True` if every row of the matrix sums to one. We use `sympy.simplify` on the difference so that the check works even when the entries are symbolic.

`markov.n_step_distribution(initial_distribution, transition_matrix, number_of_steps)` multiplies the row vector π_0 by P^n. The matrix power is computed by `sympy` using fast exponentiation, so even moderately large values of n are cheap.

`markov.stationary_distribution(transition_matrix)` returns the row vector π with $\pi P = \pi$ and $\sum_i \pi_i = 1$. We compute it as the null space of $P^T - I$ and normalise. This direct approach is short and works on any matrix; for very large chains a sparse implementation would be more efficient, but the library is designed for small chains where exactness matters more than speed.

3 Worked example

We illustrate the library on the two-state chain with transition matrix

$$P = \begin{pmatrix} 7/10 & 3/10 \\ 4/10 & 6/10 \end{pmatrix}.$$

The stationary distribution can be computed by hand: solving $\pi P = \pi$ with $\pi_0 + \pi_1 = 1$ gives $\pi = (4/7, 3/7)$. The library returns this answer exactly. Starting from $\pi_0 = (1, 0)$, the distributions after 1, 2 and 50 steps are shown in Table 1.

n	π_n
0	$(1, 0)$
1	$(7/10, 3/10)$
2	$(61/100, 39/100)$
50	numerically $(0.571429, 0.428571)$

Table 1: **Convergence to the stationary distribution.** Starting from $\pi_0 = (1, 0)$, the distribution after n steps moves quickly towards the stationary distribution $(4/7, 3/7) = (0.571\ldots, 0.428\ldots)$. After 50 steps the error in each component is below 10^{-10}.

4 Discussion

For small chains the stationary distribution often has a clean rational form (such as $(4/7, 3/7)$) that a numeric library would round to a decimal. We wanted to recover the closed form, and `sympy.Matrix` keeps every step exact.

The library handles only finite chains, and only the discrete-time case. A natural extension is to add continuous-time chains, where the rate matrix Q plays the role of $P - I$ and the distribution after time t is $\pi_0 \exp(Qt)$. Symbolic matrix exponentials are available in `sympy` so this would follow the same shape as the existing code.

5 Conclusion

We have presented `markov`, a three-function library for finite discrete-time Markov chains, written on top of `sympy.Matrix`. The library recovers the stationary distribution of a two-state chain in closed form and the iterates converge to it as expected.

References

[1] Vincent Knight. Python for Mathematics, 2024.

[2] Aaron Meurer, Christopher P. Smith, Mateusz Paprocki, Ondřej Čertík, Sergey B. Kirpichev, Matthew Rocklin, AmiT Kumar, Sergiu Ivanov, Jason K. Moore, Sartaj Singh, et al. SymPy: symbolic computing in Python. *PeerJ Computer Science*, 3:e103, 2017.

[3] James R. Norris. *Markov Chains*. Cambridge University Press, 1998.

[4] Daniele Procida. Diátaxis: a systematic framework for technical documentation authoring, 2017.