ChainoPy: A Python Library for Discrete Time Markov Chain Based Stochastic Analysis

Summmary

Modeling time series data, such as stock prices and text sequences, is effectively achieved using Markov Chains. ChainoPy facilitates the modeling of time series data with Markov Chains and Markov Switching Models, optimizing for computational efficiency in terms of speed and memory usage. Additionally, ChainoPy enables the integration of probabilistic models like Markov Chains with Neural Networks, traditionally considered deterministic, through the MarkovChainNeuralNetwork class. This hybrid approach leverages the strengths of both probabilistic and neural network methodologies.

Statement of Need

There are limitations in current Markov Chain packages like PyDTMC (Belluzzo 2024), simple-markov (Charisopoulos and Andrikopoulos 2016), mchmm (Terpilovskii 2021) that rely solely on NumPy (Harris et al. 2020) and Python for implementation. Markov Chains often require iterative convergence-based algorithms (Rosenthal 1995), where Python's dynamic typing, Global Interpreter Lock (GIL), and garbage collection can hinder potential performance improvements like parallelism. To address these issues, we enhance our library with extensions like Cython for efficient algorithm implementation. Additionally, we introduce a Markov Chain Neural Network (Awiszus and Rosenhahn 2018) that simulates given Markov Chains while preserving statistical properties from the training data. This approach eliminates the need for post-processing steps such as sampling from the outcome distribution while giving neural networks stochastic properties rather than deterministic behavior. Finally, we implement the famous Markov Switching Models (Hamilton 2010) which are one of the fundamental and widely used models in applications such as Stock Market price prediction. ChainoPy enables new workflows through its advanced algorithms, such as Markov Chain Neural Networks and Markov Switching Models, which are not available in PyDTMC. These capabilities, combined with significant performance improvements in both fast and slow functions, provide added value for complex stochastic analysis tasks.

Implementation

We implement three public classes `MarkovChain`, `MarkovChainNeuralNetwork` and `MarkovSwitchingModel` that contain core functionalities of the package. Performance intensive functions for the `MarkovChain` class are implemented in the `_backend` directory where a custom Cython (Behnel et al. 2010) backend is implemented circumventing drawbacks of Python like the GIL, dynamic typing, etc. The `MarkovChain` class implements various functionalities for discrete-time Markov chains. It provides methods for fitting the transition matrix from data, simulating the chain, and calculating properties. It also supports visualization for Markov chains.

We do the following key optimizations:

- Efficient matrix power: If the matrix is diagonalizable, an eigenvalue decomposition based matrix power is performed.
- Parallel Execution: Some functions are parallelized.
- `__slots__` usage: `__slots__` is used instead of `__dict__` for storing object attributes, reducing memory overhead.
- Caching decorator: Class methods are decorated with caching to avoid recomputation of unnecessary results.
- Direct LAPACK use: LAPACK function `dgeev` is directly used to calculate stationary-distribution via SciPy's (Virtanen et al. 2020) `cython_lapack` API instead of additional NumPy overhead.
- Utility functions for visualization: Utility functions are implemented for visualizing the Markov chain.
- Sparse storage of transition matrix: The model is stored as a JSON object, and if 40% or more elements of the transition matrix are near zero, it is stored in a sparse format.

The `MarkovChainNeuralNetwork` implementation defines a neural network model, using PyTorch (Ansel et al. 2024) for simulating Markov chain behavior. It takes a Markov chain object and the number of layers as input, with each layer being a linear layer. The model's forward method computes the output probabilities for the next state. The model is trained using stochastic gradient descent (SGD) with a learning rate scheduler. Finally, the model's performance is evaluated using the Kullback–Leibler divergence between the original Markov chain's transition probabilities and those estimated from the simulated walks.

Documentation, Testing and Benchmarking

For documentation we use Sphinx. For yesting and benchmarking we use the Pytest and PyDTMC (Belluzzo 2024) packages.

The results are as follows:

- `is_absorbing` Methods

Transition-Matrix Size	10		50		100	
	Mean	St. dev	Mean	St. dev	Mean	St. dev
Function						
1. is_absorbing (ChainoPy)	97.3ns	2.46ns	91.8ns	0.329ns	98ns	0.4ns
1. is_absorbing (PyDTMC)	386ns	5.79ns	402ns	2.01ns	417ns	3ns

- `stationary_dist` vs `pi` Methods

Transition-Matrix Size	10		50		100	
	Mean	St. dev	Mean	St. dev	Mean	St. dev
Function						
1. stationary_dist (ChainoPy)	1.47us	1.36us	93.4ns	5.26ns	96.6ns	3.9ns
1. pi (PyDTMC)	137us	12.9us	395ns	15.4ns	398ns	10.5ns

- `fit` vs `fit_sequence` Method:

Number of Words	10		50		100	
	Mean	St. dev	Mean	St. dev	Mean	St. dev
Function						
1. fit (ChainoPy)	116 µs	5.28 µs	266 µs	15 µs	496 µs	47.3 µs
1. fit_sequence (PyDTMC)	14 ms	1.74 ms	14.4 ms	1.17 ms	17.3 ms	2.18 ms

- `simulate` Method

Transition-Matrix Size	N-Steps	ChainoPy Mean	ChainoPy St. dev	PyDTMC Mean	PyDTMC St. dev
10	1000	22.8 ms	2.32 ms	28.2 ms	933 µs
	5000	86.8 ms	2.76 ms	155 ms	5.25 ms
50	1000	17.6 ms	1.2 ms	29.9 ms	1.09 ms
	5000	84.5 ms	4.84 ms	161 ms	7.62 ms
100	1000	21.6 ms	901 µs	37.4 ms	3.99 ms
	5000	110 ms	11.3 ms	162 ms	5.75 ms
500	1000	24 ms	3.73 ms	39.6 ms	6.07 ms
	5000	112 ms	6.63 ms	178 ms	26.5 ms
1000	1000	26.1 ms	620 µs	46.1 ms	6.47 ms
	5000	136 ms	2.49 ms	188 ms	2.43 ms
2500	1000	42 ms	3.77 ms	59.6 ms	2.29 ms
	5000	209 ms	16.4 ms	285 ms	27.6ms

Apart from this, we test the `MarkovChainNeuralNetworks` by training them and comparing random walks between the original `MarkovChain` object and those generated by `MarkovChainNeuralNetworks` through a histogram.

Conclusion

In conclusion, ChainoPy offers a Python library for discrete-time Markov Chains and includes features for Markov Chain Neural Networks, providing a useful tool for researchers and practitioners in stochastic analysis with efficient performance.

References

Ansel, Jason, Edward Yang, Horace He, Natalia Gimelshein, Animesh Jain, Michael Voznesensky, Bin Bao, et al. 2024. "PyTorch 2: Faster Machine Learning Through Dynamic Python Bytecode Transformation and Graph Compilation." In *Proceedings of the 29th Acm International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, 929–47. https://doi.org/10.1145/3620665.3640366.

Awiszus, Maren, and Bodo Rosenhahn. 2018. "Markov Chain Neural Networks." In *Proceedings of the Ieee Conference on Computer Vision and Pattern Recognition Workshops*, 2180–7. https://doi.org/10.1109/CVPRW.2018.00293.

Behnel, Stefan, Robert Bradshaw, Craig Citro, Lisandro Dalcin, Dag Sverre Seljebotn, and Kurt Smith. 2010. "Cython: The Best of Both Worlds." *Computing in Science & Engineering* 13 (2): 31–39. https://doi.org/10.1109/MCSE.2010.118.

Belluzzo, Tommaso. 2024. "PyDTMC." https://github.com/TommasoBelluzzo/PyDTMC. https://github.com/TommasoBelluzzo/PyDTMC.

Charisopoulos, Vasilis, and Kostis Andrikopoulos. 2016. "Simple-Markov." https://github.com/Mandragorian/simple-markov. https://github.com/Mandragorian/simple-markov.

Hamilton, James D. 2010. "Regime Switching Models." In *Macroeconometrics and Time Series Analysis*, 202–9. Springer. https://doi.org/10.1057/9780230280830_23.

Harris, Charles R, K Jarrod Millman, Stéfan J Van Der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, et al. 2020. "Array Programming with Numpy." *Nature* 585 (7825): 357–62. https://doi.org/10.1038/s41586-020-2649-2.

Rosenthal, Jeffrey S. 1995. "Convergence Rates for Markov Chains." *SIAM Review* 37 (3): 387–405. https://doi.org/10.1137/1037083.

Terpilovskii, Maksim. 2021. "Mchmm." https://github.com/maximtrp/mchmm. https://github.com/maximtrp/mchmm.

Virtanen, Pauli, Ralf Gommers, Travis E Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, et al. 2020. "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python." *Nature Methods* 17 (3): 261–72. https://doi.org/10.1038/s41592-019-0686-2.