PolytopeWalk: Sparse MCMC Sampling over Polytopes

Benny Sun[1] **and Yuansi Chen**[2]

1 Department of Statistics, Duke University **2** Department of Mathematics, ETH Zurich

DOI: 10.21105/joss.07957

Software

- Review
- Repository
- Archive

Editor: Vissarion Fisikopoulos

Reviewers:

- @martinmodrak
- @matt-graham

Submitted: 04 March 2025
Published: 22 July 2025

Summary

High dimensional sampling is an important computational tool in statistics, with applications in stochastic simulation, volume computation, and fast randomized algorithms. We present `PolytopeWalk`, a scalable library designed for sampling from a uniform distribution over polytopes, which are bounded geometric objects formed by linear inequalities. For sampling, we use Markov chain Monte Carlo (MCMC) methods, defined as a family of algorithms for generating approximate samples from a target probability distribution. Six state-of-the-art MCMC algorithms are implemented, including the Dikin, Vaidya, and John Walk. Additionally, we introduce novel sparse constrained formulations of these algorithms, enabling efficient sampling from sparse polytopes of the form $\mathcal{K}_2 = \{x \in \mathbb{R}^d \mid Ax = b, x \succeq_k 0\}$. This implementation maintains sparsity in A, ensuring scalability to higher dimensional settings in per-iteration cost. Finally, `PolytopeWalk` includes implementations of 2 preprocessing algorithms, facial reduction and initialization, thus providing an end-to-end solution.

Statement of Need

High dimensional sampling is a fundamental problem in many computational disciplines such as statistics, probability, and operation research. For example, sampling is applied in portfolio optimization (Calès et al., 2023), metabolic networks in biology (Heirendt et al., 2018) and volume approximation over convex shapes (Simonovits, 2003). Markov chain Monte Carlo (MCMC) sampling algorithms offer a natural and scalable solution to this problem. These algorithms construct a Markov chain whose stationary distribution matches the target distribution. By running the chain for a large number of steps to ensure mixing, MCMC algorithms can efficiently generate approximately independent samples close to the target distribution, while not suffering from the curse of dimension issues.

This package focuses on sampling from a uniform distribution over a user-specified polytope. We define the polytope as the following. Let $A \in \mathbb{R}^{n\times d}$, $b \in \mathbb{R}^n$ and let $x \succeq_k y$ mean that the last k-coordinates of x are greater than or equal to the corresponding coordinates of y, i.e., $\{x_{d-k+1} - y_{d-k+1} \geq 0, ..., x_d - y_d \geq 0\}$. Depending on whether we allow equality constraints, the sampling problem can be formalized in two forms:

1. The full-dimensional form:

$$\mathcal{K}_1 = \{x \in \mathbb{R}^d \mid Ax \leq b\}, \tag{1}$$

   where $\mathcal{K}_1$ is specified via n inequality constraints.

2. The constrained form:

$$\mathcal{K}_2 = \{x \in \mathbb{R}^d \mid Ax = b, x \succeq_k 0\}, \tag{2}$$

   where $\mathcal{K}_2$ is specified via n equality constraints and k coordinate inequality constraints.

Large polytopes with sparse constraints are common in many applications (Kook et al., 2022). The largest human metabolic network RECON3D is modeled as a 13543-dimensional sparse polytope (King et al., 2015). Moreover, linear programming datasets from `NetLib` are naturally in the constrained form, where A matrix is sparse. These applications motivate the need for MCMC algorithms that leverage $\mathcal{K}_2$ form. We implement novel interior-point-method-based MCMC algorithms optimized for large and sparse constrained polytopes. By exploiting sparsity, our algorithms scale well in per-iteration cost as a function of increasing dimension. Using the Dikin Walk, we can perform over 300 steps per second for a 10^4 dimensional simplex. For reference, a heuristic for generating 1 sample is 100 steps times the mixing time.

Interior-point-method-based MCMC sampling algorithms on a polytope are modifications of the Ball Walk (Vempala, 2005), incorporating key concepts from interior-point methods in optimization. These algorithms operate in two primary steps. First, the algorithm generates a proposal distribution whose covariance matrix is state-dependent and equal to the inverse of the Hessian matrix of a specified barrier function, capturing the local geometry of the polytope. Second, the algorithm employs the Metropolis-Hastings accept-reject step to ensure that its stationary distribution is uniform on the polytope (Hastings, 1970; Metropolis et al., 1953). Using a state-dependent proposal distribution that adapts to the polytope's local geometry, these MCMC algorithms achieve an improved mixing rate.

In `PolytopeWalk`, we implement 4 barrier MCMC sampling algorithms and 2 standard random walk algorithms (6 total) in the sparse-constrained and full-dimensional formulation. `PolytopeWalk` makes meaningful strides in the open-source development of MCMC, speeding up calculations for sparse high-dimensional sampling. Finally, we provide an implementation of the Facial Reduction algorithm, described in detail in the Preprocessing Algorithms section. Additional technical details can be found in a separate paper (Sun & Chen, 2024).

Package Overview

`PolytopeWalk` is an open-source library written in C++ with Python wrapper code, providing accelerated MCMC sampling algorithms in both $\mathcal{K}_1$ and $\mathcal{K}_2$ formulation. Source code is written with `Eigen` for linear algebra (Guennebaud et al., 2010), `glpk` for linear programming (Makhorin, 2012), and `pybind` for Python binding (Jakob et al., 2017). In Python, `PolytopeWalk` relies on NumPy (Harris et al., 2020) and SciPy (Virtanen et al., 2020).

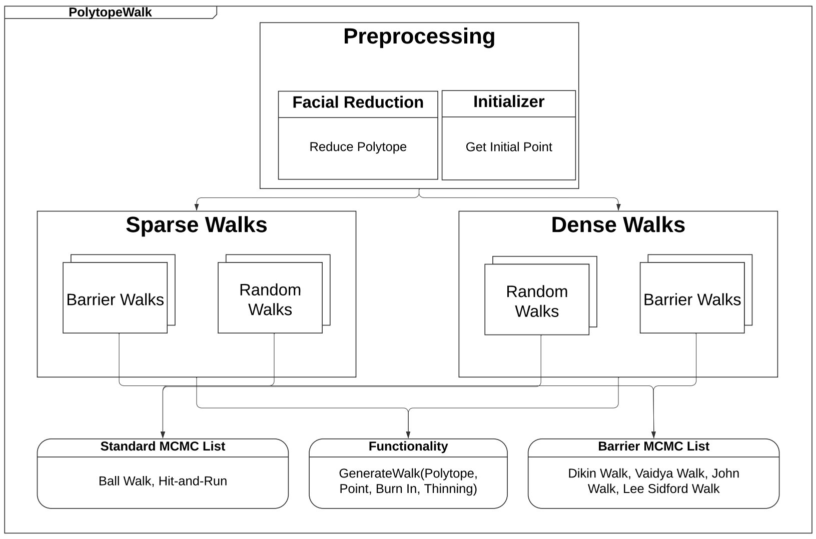

Figure 1: Code Structure of Package

Random Walk Algorithms

Mixing times refer to the required number of steps to converge to stationary distribution. In each, d refers to the dimension of the polytope and n refers to the number of boundaries ($\mathcal{K}_1$ dimensions). In the first 2 walks, R^2/r^2 means where the convex body contains a ball of radius r and is mostly contained in a ball of radius R.

Name	Mixing Time	Author
`Ball Walk`	$O(d^2R^2/r^2)$	Vempala (2005)
`Hit and Run`	$O(d^2R^2/r^2)$	Lovasz (1999)
`Dikin Walk`	$O(nd)$	Sachdeva et al. (2015)
`Vaidya Walk`	$O(n^{1/2}d^{3/2})$	Chen et al. (2018)
`John Walk`	$O(d^{2.5})$	Chen et al. (2018)
`Lee Sidford Walk`	$O(d^2)$	Laddha et al. (2019)

Preprocessing Algorithms

`PolytopeWalk` comes with 2 preprocessing algorithms: initialization and facial reduction.

Initialization: If the user cannot specify a point inside of the polytope to start, `PolytopeWalk` provides a class to compute an initial point well within the polytope for both the full-dimensional formulation and constrained formulation.

Facial Reduction: We adopt the facial reduction algorithm implementation from Drusvyatskiy's research (Drusvyatskiy & Wolkowicz, 2017; Im & Wolkowicz, 2023). In the constrained formulation $\mathcal{K}_2 = \{x \in \mathbb{R}^d \mid Ax = b, x \succeq_k 0\}$, degeneracy occurs when there is a lack of strict feasibility in the polytope: there does not exist an $x \in \mathbb{R}^d$ such that $Ax = b$ and $x \succ_k 0$. Thus, degeneracy exists in polytopes when the lower-dimensional polytope is embedded in a higher dimension. The facial reduction algorithm eliminates variables in the last k dimensions fixed at 0, thus ensuring numerical stability for sampling.

Package Comparison

Feature	`PolytopeWalk`	`Volesti`	`PolytopeSampler`	`Polyrun`
Constrained Formulation	Y	N	Y	Y
Sparse Friendly	Y	N	Y	N
C++ Implementation	Y	Y	Y	N
Facial Reduction	Y	N	N	N
Dikin Walk	Y	Y	N	N
Vaidya Walk	Y	Y	N	N
John Walk	Y	Y	N	N
Lee-Sidford Walk	Y	N	N	N

Table II contrasts the features of `PolytopeWalk` with `Volesti` (Chalkis et al., 2025), `PolytopeSampler` (Kook et al., 2022), and `Polyrun` (Ciomek & Kadziński, 2021). `Volesti` is implemented in C++ with some of its code represented in the Python library `Dingo` (Chalkis et al., 2024). `PolytopeSampler` only works on Matlab and `Polyrun` on Java. Thus, `PolytopeWalk` adds additional features and novelties not found in other MCMC sampling packages.

Acknowledgements

Much work was done while Yuansi Chen was an assistant professor in the Department of Statistical Science at Duke University. Both authors are partially supported by NSF CAREER Award DMS-2237322, Sloan Research Fellowship and Ralph E. Powe Junior Faculty Enhancement Awards.

References

Calès, L., Chalkis, A., Emiris, I. Z., & Fisikopoulos, V. (2023). Practical volume approximation of high-dimensional convex bodies, applied to modeling portfolio dependencies and financial crises. *Computational Geometry*, *109*, 101916. https://doi.org/10.1016/j.comgeo.2022.101916

Chalkis, A., Fisikopoulos, V., Papachristou, M., & Tsigaridas, E. (2025). Volesti: A C++ library for sampling and volume computation on convex bodies. *Journal of Open Source Software*, *10*(108), 7886. https://doi.org/10.21105/joss.07886

Chalkis, A., Fisikopoulos, V., Tsigaridas, E., & Zafeiropoulos, H. (2024). Dingo: A Python package for metabolic flux sampling. *Bioinformatics Advances*, *4*(1), vbae037. https://doi.org/10.1093/bioadv/vbae037

Ciomek, K., & Kadziński, M. (2021). Polyrun: A Java library for sampling from the bounded convex polytopes. *SoftwareX*, *13*, 100659. https://doi.org/10.1016/j.softx.2021.100659

Drusvyatskiy, D., & Wolkowicz, H. (2017). The many faces of degeneracy in conic optimization. *Foundations and Trends® in Optimization*, *3*(2), 77–170. https://doi.org/10.1561/2400000011

Guennebaud, G., Jacob, B., & others. (2010). *Eigen v3*. http://eigen.tuxfamily.org.

Harris, C. R., Millman, K. J., Walt, S. J. van der, Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., Kerkwijk, M. H. van, Brett, M., Haldane, A., Río, J. F. del, Wiebe, M., Peterson, P., … Oliphant, T. E. (2020). Array programming with NumPy. *Nature*, *585*(7825), 357–362. https://doi.org/10.1038/s41586-020-2649-2

Hastings, W. K. (1970). Monte Carlo sampling methods using Markov chains and their applications. *Biometrika*, *57*(1), 97–109. https://doi.org/10.1093/biomet/57.1.97

Heirendt, L., Arreckx, S., Pfau, T., Mendoza, S. N., Richelle, A., Heinken, A., Haraldsdóttir, H. S., Wachowiak, J., Keating, S. M., Vlasov, V., Magnusdóttir, S., Ng, C. Y., Preciat, G., Žagare, A., Chan, S. H. J., Aurich, M. K., Clancy, C. M., Modamio, J., Sauls, J. T., … Fleming, R. M. T. (2018). *Creation and analysis of biochemical constraint-based models: The COBRA toolbox v3.0*. https://arxiv.org/abs/1710.04038

Im, H., & Wolkowicz, H. (2023). Revisiting degeneracy, strict feasibility, stability, in linear programming. *European Journal of Operational Research*, *310*(2), 495–510. https://doi.org/10.1016/j.ejor.2023.03.021

Jakob, W., Rhinelander, J., & Moldovan, D. (2017). *pybind11 – Seamless operability between C++11 and Python*.

King, Z. A., Lu, J., Dräger, A., Miller, P., Federowicz, S., Lerman, J. A., Ebrahim, A., Palsson, B. O., & Lewis, N. E. (2015). BiGG Models: A platform for integrating, standardizing and sharing genome-scale models. *Nucleic Acids Research*, *44*(D1), D515–D522. https://doi.org/10.1093/nar/gkv1049

Kook, Y., Lee, Y. T., Shen, R., & Vempala, S. S. (2022). Sampling with riemannian hamiltonian monte carlo in a constrained space. *Proceedings of the 36th International Conference on*

Neural Information Processing Systems. ISBN: 9781713871088

Makhorin, A. (2012). *(GNU linear programming kit) package*.

Metropolis, N., Rosenbluth, A. W., Rosenbluth, M. N., Teller, A. H., & Teller, E. (1953). Equation of state calculations by fast computing machines. *The Journal of Chemical Physics*, *21*(6), 1087–1092. https://doi.org/10.1063/1.1699114

Simonovits, M. (2003). How to compute the volume in high dimension? *Mathematical Programming*, *97*. https://doi.org/10.1007/s10107-003-0447-x

Sun, B., & Chen, Y. (2024). *PolytopeWalk: Sparse MCMC Sampling over Polytopes*. https://arxiv.org/abs/2412.06629

Vempala, S. (2005). Geometric random walks: A survey. *Combinatorial and Computational Geometry*, *52*. https://doi.org/10.1017/9781009701259.033

Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., van der Walt, S. J., Brett, M., Wilson, J., Millman, K. J., Mayorov, N., Nelson, A. R. J., Jones, E., Kern, R., Larson, E., … SciPy 1.0 Contributors. (2020). SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, *17*, 261–272. https://doi.org/10.1038/s41592-019-0686-2