Bayesian Analysis (2004) **1**, Number 1

# Fast Simulation of Hyperplane-Truncated Multivariate Normal Distributions

Yulai Cong*, Bo Chen†§, and Mingyuan Zhou‡§

**Abstract.** We introduce a fast and easy-to-implement simulation algorithm for a multivariate normal distribution truncated on the intersection of a set of hyperplanes, and further generalize it to efficiently simulate random variables from a multivariate normal distribution whose covariance (precision) matrix can be decomposed as a positive-definite matrix minus (plus) a low-rank symmetric matrix. Example results illustrate the correctness and efficiency of the proposed simulation algorithms.

**Keywords:** Cholesky decomposition, conditional distribution, equality constraints, high-dimensional regression, structured covariance/precision matrix

## 1 Introduction

We investigate the problem of simulation from a multivariate normal (MVN) distribution whose samples are restricted to the intersection of a set of hyperplanes, which is shown to be inherently related to the simulation of a conditional distribution of a MVN distribution. A naive approach, which linearly transforms a random variable drawn from the conditional distribution of a related MVN distribution, requires a large number of intermediate variables that are often computationally expensive to instantiate. To address this issue, we propose a fast and exact simulation algorithm that directly projects a MVN random variable onto the intersection of a set of hyperplanes. We further show that sampling from a MVN distribution, whose covariance (precision) matrix can be decomposed as the sum (difference) of a positive-definite matrix, whose inversion is known or easy to compute, and a low-rank symmetric matrix, may also be made significantly fast by exploiting this newly proposed stimulation algorithm for hyperplane-truncated MVN distributions, avoiding the need of Cholesky decomposition that has a computational complexity of $O(k^3)$ (Golub and Van Loan 2012), where $k$ is the dimension of the MVN random variable.

Related to the problems under study, the simulation of MVN random variables subject to certain constraints (Gelfand et al. 1992) has been investigated in many other different settings, such as multinomial probit and logit models (Albert and Chib 1993;

*National Laboratory of Radar Signal Processing and Collaborative Innovation Center of Information Sensing & Understanding, Xidian University, Xi'an, Shaanxi 710071, China, yulai_cong@163.com

†National Laboratory of Radar Signal Processing and Collaborative Innovation Center of Information Sensing & Understanding, Xidian University, Xi'an, Shaanxi 710071, China, bchen@mail.xidian.edu.cn

‡McCombs School of Business, The University of Texas at Austin, Austin, TX 78712, USA, mingyuan.zhou@mccombs.utexas.edu

§Corresponding authors.

McCulloch et al. 2000; Imai and van Dyk 2005; Train 2009; Holmes and Held 2006; Johndrow et al. 2013), Bayesian isotonic regression (Neelon and Dunson 2004), Bayesian bridge (Polson et al. 2014), blind source separation (Schmidt 2009), and unmixing of hyperspectral data (Altmann et al. 2014; Dobigeon et al. 2009a). A typical example arising in these different settings is to sample a random vector $\boldsymbol{x} \in \mathbb{R}^k$ from a MVN distribution subject to $k$ inequality constraints as

$$\boldsymbol{x} \sim \mathcal{N}_{\mathcal{S}}(\boldsymbol{\mu}, \boldsymbol{\Sigma}), \quad \mathcal{S} = \{\boldsymbol{x} : \boldsymbol{l} \leq \mathbf{G}\boldsymbol{x} \leq \boldsymbol{u}\}, \tag{1}$$

where $\mathcal{N}_{\mathcal{S}}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ represents a MVN distribution truncated on the sample space $\mathcal{S}$, $\boldsymbol{\mu} \in \mathbb{R}^k$ is the mean, $\boldsymbol{\Sigma} \in \mathbb{R}^{k \times k}$ is the covariance matrix, $\mathbf{G} \in \mathbb{R}^{k \times k}$ is a full-rank matrix, $\boldsymbol{l} \in \mathbb{R}^k$, $\boldsymbol{u} \in \mathbb{R}^k$, and $\boldsymbol{l} < \boldsymbol{u}$. If the elements of $\boldsymbol{l}$ and $\boldsymbol{u}$ are permitted to be $-\infty$ and $+\infty$, respectively, then both single sided and fewer than $k$ inequality constraints are allowed. Equivalently, as in Geweke (1991, 1996), one may let $\boldsymbol{x} = \boldsymbol{\mu} + \mathbf{G}^{-1}\boldsymbol{z}$ and use Gibbs sampling (Geman and Geman 1984; Gelfand and Smith 1990) to sample the $k$ elements of $\boldsymbol{z}$ one at a time conditioning on all the others from a univariate truncated normal distribution, for which efficient algorithms exist (Robert 1995; Damien and Walker 2001; Chopin 2011). To deal with the case that the number of linear constraints imposed on $\boldsymbol{x}$ exceed its dimension $k$ and to obtain better mixing, one may consider the Gibbs sampling algorithm for truncated MVN distributions proposed in Rodriguez-Yam et al. (2004). In addition to Gibbs sampling, to sample truncated MVN random variables, one may also consider Hamiltonian Monte Carlo (Pakman and Paninski 2014; Lan et al. 2014) and a minimax tilting method proposed in Botev (2016).

## 2 Hyperplane-truncated and conditional MVNs

For the problem under study, we express a $k$-dimensional MVN distribution truncated on the intersection of $k_2 < k$ hyperplanes as

$$\boldsymbol{x} \sim \mathcal{N}_{\mathcal{S}}(\boldsymbol{\mu}, \boldsymbol{\Sigma}), \quad \mathcal{S} = \{\boldsymbol{x} : \mathbf{G}\boldsymbol{x} = \boldsymbol{r}\}, \tag{2}$$

where

$$\mathbf{G} \in \mathbb{R}^{k_2 \times k}, \quad \boldsymbol{r} \in \mathbb{R}^{k_2},$$

and $\text{Rank}(\mathbf{G}) = k_2$. The probability density function can be expressed as

$$p(\boldsymbol{x} \,|\, \boldsymbol{\mu}, \boldsymbol{\Sigma}, \mathbf{G}, \boldsymbol{r}) = \frac{1}{Z} \exp\left[-\frac{1}{2}(\boldsymbol{x} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1} (\boldsymbol{x} - \boldsymbol{\mu})\right] \delta(\mathbf{G}\boldsymbol{x} = \boldsymbol{r}), \tag{3}$$

where $Z$ is a constant ensuring $\int p(\boldsymbol{x} \,|\, \boldsymbol{\mu}, \boldsymbol{\Sigma}, \mathbf{G}, \boldsymbol{r}) d\boldsymbol{x} = 1$, and $\delta(x) = 1$ if the condition $x$ is satisfied and $\delta(x) = 0$ otherwise. Let us partition $\mathbf{G}$, $\boldsymbol{x}$, $\boldsymbol{\mu}$, $\boldsymbol{\Sigma}$, and $\boldsymbol{\Lambda} = \boldsymbol{\Sigma}^{-1}$ as

$$\mathbf{G} = (\mathbf{G}_1, \mathbf{G}_2), \; \boldsymbol{x} = \begin{bmatrix} \boldsymbol{x}_1 \\ \boldsymbol{x}_2 \end{bmatrix}, \; \boldsymbol{\mu} = \begin{bmatrix} \boldsymbol{\mu}_1 \\ \boldsymbol{\mu}_2 \end{bmatrix}, \; \boldsymbol{\Sigma} = \begin{bmatrix} \boldsymbol{\Sigma}_{11} & \boldsymbol{\Sigma}_{12} \\ \boldsymbol{\Sigma}_{21} & \boldsymbol{\Sigma}_{22} \end{bmatrix}, \text{ and } \boldsymbol{\Lambda} = \begin{bmatrix} \boldsymbol{\Lambda}_{11} & \boldsymbol{\Lambda}_{12} \\ \boldsymbol{\Lambda}_{21} & \boldsymbol{\Lambda}_{22} \end{bmatrix},$$

whose sizes are

$$(k_2 \times k_1, k_2 \times k_2), \; \begin{bmatrix} k_1 \times 1 \\ k_2 \times 1 \end{bmatrix}, \; \begin{bmatrix} k_1 \times 1 \\ k_2 \times 1 \end{bmatrix}, \; \begin{bmatrix} k_1 \times k_1 & k_1 \times k_2 \\ k_2 \times k_1 & k_2 \times k_2 \end{bmatrix}, \text{ and } \begin{bmatrix} k_1 \times k_1 & k_1 \times k_2 \\ k_2 \times k_1 & k_2 \times k_2 \end{bmatrix},$$

respectively, where $k = k_1 + k_2$, $\mathbf{\Sigma}_{21} = \mathbf{\Sigma}_{12}^T$, and $\mathbf{\Lambda}_{21} = \mathbf{\Lambda}_{12}^T$.

A special case that frequently arises in real applications is when $\mathbf{G}_1 = \mathbf{0}_{k_2 \times k_1}$ and $\mathbf{G}_2 = \mathbf{I}_{k_2}$, which means $(\mathbf{0}_{k_2 \times k_1}, \mathbf{I}_{k_2})\boldsymbol{x} = \boldsymbol{x}_2 = \boldsymbol{r}$ and the need is to simulate $\boldsymbol{x}_1$ given $\boldsymbol{x}_2 = \boldsymbol{r}$. For a MVN random variable $\boldsymbol{x} \sim \mathcal{N}(\boldsymbol{\mu}, \mathbf{\Sigma})$, it is well known, *e.g.*, in Tong (2012), that the conditional distribution of $\boldsymbol{x}_1$ given $\boldsymbol{x}_2 = \boldsymbol{r}$, *i.e.*, the distribution of $\boldsymbol{x}$ restricted to $\mathcal{S} = \{\boldsymbol{x} : (\mathbf{0}_{k_2 \times k_1}, \mathbf{I}_{k_2})\boldsymbol{x} = \boldsymbol{r}\}$, can be expressed as

$$\boldsymbol{x}_1 \,|\, \boldsymbol{x}_2 = \boldsymbol{r} \;\sim\; \mathcal{N}\left[\boldsymbol{\mu}_1 + \mathbf{\Sigma}_{12}\mathbf{\Sigma}_{22}^{-1}(\boldsymbol{r} - \boldsymbol{\mu}_2),\;\; \mathbf{\Sigma}_{11} - \mathbf{\Sigma}_{12}\mathbf{\Sigma}_{22}^{-1}\mathbf{\Sigma}_{21}\right]. \tag{4}$$

Alternatively, applying the Woodbury matrix identity to relate the entries of the covariance matrix $\mathbf{\Sigma}$ to those of the precision matrix $\mathbf{\Lambda}$, one may obtain the following equivalent expression as

$$\boldsymbol{x}_1 \,|\, \boldsymbol{x}_2 = \boldsymbol{r} \;\sim\; \mathcal{N}\left[\boldsymbol{\mu}_1 - \mathbf{\Lambda}_{11}^{-1}\mathbf{\Lambda}_{12}(\boldsymbol{r} - \boldsymbol{\mu}_2),\; \mathbf{\Lambda}_{11}^{-1}\right]. \tag{5}$$

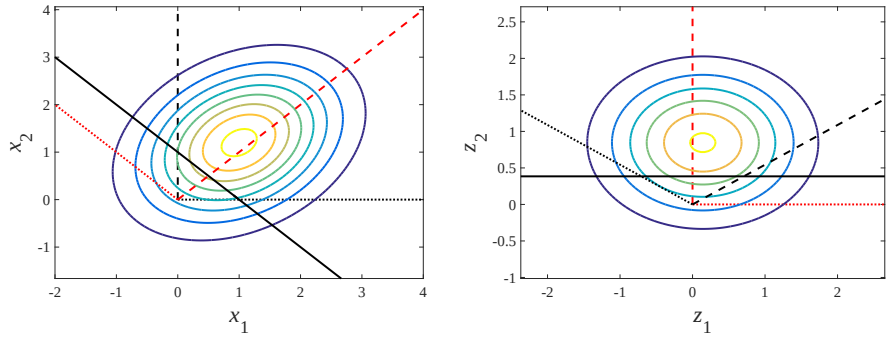
In a general setting where $\mathbf{G} \neq (\mathbf{0}_{k_2 \times k_1}, \mathbf{I}_{k_2})$, let us define a full rank linear transformation matrix $\mathbf{H} \in \mathbb{R}^{k \times k}$, with $(\mathbf{H}_1, \mathbf{H}_2)$ as the $(k \times k_1, k \times k_2)$ partition of $\mathbf{H}$, where the columns of $\mathbf{H}_1 \in \mathbb{R}^{k \times k_1}$ span the null space of the $k_2$ rows of $\mathbf{G}$, making $\mathbf{GH} = (\mathbf{GH}_1, \mathbf{GH}_2) = (\mathbf{0}_{k_2 \times k_1}, \mathbf{GH}_2)$, where $\mathbf{GH}_2$ is a $k_2 \times k_2$ full rank matrix. For example, a linear transformation matrix $\mathbf{H}$ that makes $\mathbf{GH} = (\mathbf{0}_{k_2 \times k_1}, \mathbf{I}_{k_2})$ can be constructed using the command $\mathbf{H} = \text{inv}([\text{null}(\mathbf{G})'; \mathbf{G}])$ in Matlab and $\mathbf{H} <-\text{solve}(\text{rbind}(\text{t}(\text{Null}(\text{t}(\mathbf{G}))), \mathbf{G}))$ in R. With $\mathbf{H}$ and $\mathbf{H}^{-1}$, one may re-express the constraints as $\mathcal{S} = \{\boldsymbol{x} : (\mathbf{0}_{k_2 \times k_1}, \mathbf{GH}_2)(\mathbf{H}^{-1}\boldsymbol{x}) = \boldsymbol{r}\}$. Denote $\boldsymbol{z} = \mathbf{H}^{-1}\boldsymbol{x}$, then we can generate $\boldsymbol{x}$ by letting $\boldsymbol{x} = \mathbf{H}\boldsymbol{z}$, where

$$\boldsymbol{z} \sim \mathcal{N}_{\mathcal{D}}[\mathbf{H}^{-1}\boldsymbol{\mu}, \mathbf{H}^{-1}\mathbf{\Sigma}(\mathbf{H}^{-1})^T], \quad \mathcal{D} = \{\boldsymbol{z} : \mathbf{GH}_2\boldsymbol{z}_2 = \boldsymbol{r}\} = \{\boldsymbol{z} : \boldsymbol{z}_2 = (\mathbf{GH}_2)^{-1}\boldsymbol{r}\}. \tag{6}$$

More specifically, denoting $\mathbf{\Lambda} = [\mathbf{H}^{-1}\mathbf{\Sigma}(\mathbf{H}^{-1})^T]^{-1} = \mathbf{H}^T\mathbf{\Sigma}^{-1}\mathbf{H}$ as the precision matrix for $\boldsymbol{z}$, we have

$$\begin{bmatrix} \mathbf{\Lambda}_{11} & \mathbf{\Lambda}_{12} \\ \mathbf{\Lambda}_{21} & \mathbf{\Lambda}_{22} \end{bmatrix} = \mathbf{H}^T\mathbf{\Sigma}^{-1}\mathbf{H} = \begin{bmatrix} \mathbf{H}_1^T\mathbf{\Sigma}^{-1}\mathbf{H}_1 & \mathbf{H}_1^T\mathbf{\Sigma}^{-1}\mathbf{H}_2 \\ \mathbf{H}_2^T\mathbf{\Sigma}^{-1}\mathbf{H}_1 & \mathbf{H}_2^T\mathbf{\Sigma}^{-1}\mathbf{H}_2 \end{bmatrix}, \tag{7}$$

and hence $\boldsymbol{x}$ truncated on $\mathcal{S}$ can be naively generated using the following algorithm, whose computational complexity is described in Table 1 of the Appendix.




(a) (b)

Figure 1: Illustration of (a) $p(\boldsymbol{x})$ in (3), where $\boldsymbol{\mu} = (1, 1.2)^T$, $\boldsymbol{\Sigma} = [(1, 0.3)^T, (0.3, 1)^T]$, $\mathbf{G} = (1, 1)$, and $\boldsymbol{r} = 1$, and (b) $p(\boldsymbol{z})$ in (6), where $\mathbf{H}_1 = (-0.7071, 0.7071)^T$, $\mathbf{H}_2 = (1.3, 1.3)^T$, and $\mathbf{H}^{-1} = [(-0.7071, 0.3846)^T, (0.7071, 0.3846)^T]$. The coordinate systems of $\boldsymbol{x}$ and $\boldsymbol{z}$ are shown in black and red, respectively, and the first and second axes of a coordinate system are shown as dotted and dashed lines, respectively.

---

**Algorithm 1** Simulation of the hyperplane truncated MVN distribution $\boldsymbol{x} \sim \mathcal{N}_{\mathcal{S}}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$, where $\mathcal{S} = \{\boldsymbol{x} : \mathbf{G}\boldsymbol{x} = \boldsymbol{r}\}$, by transforming a random variable drawn from the conditional distribution of another MVN distribution.

---

- Find $\mathbf{H} = (\mathbf{H}_1, \mathbf{H}_2)$ that satisfies $\mathbf{GH} = (\mathbf{GH}_1, \mathbf{GH}_2) = (\mathbf{0}_{k_2 \times k_1}, \mathbf{GH}_2)$, where $\mathbf{GH}_2$ is a full rank matrix;
- Let $\boldsymbol{z}_2 = (\mathbf{GH}_2)^{-1}\boldsymbol{r}$, $\boldsymbol{\Lambda}_{11} = \mathbf{H}_1^T\boldsymbol{\Sigma}^{-1}\mathbf{H}_1$, and $\boldsymbol{\Lambda}_{12} = \mathbf{H}_1^T\boldsymbol{\Sigma}^{-1}\mathbf{H}_2$;
- Sample $\boldsymbol{z}_1 \,|\, \boldsymbol{z}_2 = (\mathbf{GH}_2)^{-1}\boldsymbol{r} \sim \mathcal{N}(\boldsymbol{\mu}_{z_1}, \boldsymbol{\Lambda}_{11}^{-1})$, where

$$\boldsymbol{\mu}_{z_1} = (\mathbf{I}_{k_1}, \mathbf{0}_{k_1 \times k_2})\mathbf{H}^{-1}\boldsymbol{\mu} - \boldsymbol{\Lambda}_{11}^{-1}\boldsymbol{\Lambda}_{12}\left[(\mathbf{GH}_2)^{-1}\boldsymbol{r} - (\mathbf{0}_{k_2 \times k_1}, \mathbf{I}_{k_2})\mathbf{H}^{-1}\boldsymbol{\mu}\right];$$

- Return $\boldsymbol{x} = \mathbf{H}\boldsymbol{z} = \mathbf{H}_1\boldsymbol{z}_1 + \mathbf{H}_2(\mathbf{GH}_2)^{-1}\boldsymbol{r}$.

---

For illustration, we consider a simple 2-dimensional example with $\boldsymbol{\mu} = (1, 1.2)^T$, $\boldsymbol{\Sigma} = [(1, 0.3)^T, (0.3, 1)^T]$, $\mathbf{G} = (1, 1)$, and $\boldsymbol{r} = 1$. If we choose $\mathbf{H}_1 = (-0.7071, 0.7071)^T$ and $\mathbf{H}_2 = (1.3, 1.3)^T$, then we have $z_2 = (\mathbf{GH}_2)^{-1}\boldsymbol{r} = (2.6)^{-1} = 0.3846$, $\boldsymbol{\Lambda}_{11} = 1.4285$, and $\boldsymbol{\Lambda}_{12} = 0$; as shown in Figure 1, we may generate $\boldsymbol{x}$ using

$$\boldsymbol{x} = (-0.7071, 0.7071)^T z_1 + (1.3, 1.3)^T z_2$$

where $z_1 \sim \mathcal{N}(0.1414, 0.7)$ and $z_2 = 0.3846$.

For high dimensional problems, however, Algorithm 1 in general requires a large number of intermediate variables that could be computationally expensive to compute.

In the following discussion, we will show how to completely avoid instantiating these intermediate variables.

# 3 Fast and exact simulation of MVN distributions

Instead of using Algorithm 1, we first provide a theorem to show how to efficiently and exactly simulate from a hyperplane-truncated MVN distribution. In the Appendix, we provide two different proofs. The first proof facilitates the derivations by employing an existing algorithm of Hoffman and Ribak (1991) and Doucet (2010), which describes how to simulate from the conditional distribution of a MVN distribution shown in (4) without computing $\boldsymbol{\Sigma}_{11} - \boldsymbol{\Sigma}_{12}\boldsymbol{\Sigma}_{22}^{-1}\boldsymbol{\Sigma}_{21}$ and its Cholesky decomposition. Note it is straightforward to verify that the algorithm in Hoffman and Ribak (1991) and Doucet (2010), as shown in the Appendix, can be considered as a special case of the proposed algorithm with $\mathbf{G} = [\mathbf{0}, \mathbf{I}]$.

**Algorithm 2** Simulation of the hyperplane truncated MVN distribution $\boldsymbol{x} \sim \mathcal{N}_{\mathcal{S}}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$, where $\mathcal{S} = \{\boldsymbol{x} : \mathbf{G}\boldsymbol{x} = \boldsymbol{r}\}$, by transforming a random variable drawn from $\boldsymbol{y} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$.

- Sample $\boldsymbol{y} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$;
- Return $\boldsymbol{x} = \boldsymbol{y} + \boldsymbol{\Sigma}\mathbf{G}^T(\mathbf{G}\boldsymbol{\Sigma}\mathbf{G}^T)^{-1}(\boldsymbol{r} - \mathbf{G}\boldsymbol{y})$, which can be realized using
  - Solve $\boldsymbol{\alpha}$ such that $(\mathbf{G}\boldsymbol{\Sigma}\mathbf{G}^T)\boldsymbol{\alpha} = \boldsymbol{r} - \mathbf{G}\boldsymbol{y}$;
  - Return $\boldsymbol{x} = \boldsymbol{y} + \boldsymbol{\Sigma}\mathbf{G}^T\boldsymbol{\alpha}$.

**Theorem 1.** *Suppose $\boldsymbol{x}$ is simulated with Algorithm 2, then it is distributed as $\boldsymbol{x} \sim \mathcal{N}_{\mathcal{S}}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$, $\mathcal{S} = \{\boldsymbol{x} : \mathbf{G}\boldsymbol{x} = \boldsymbol{r}\}$, where $\mathbf{G} \in \mathbb{R}^{k_2 \times k}$, $\boldsymbol{r} \in \mathbb{R}^{k_2}$, and Rank$(\mathbf{G}) = k_2 < k$.*

The above algorithm and theorem, whose computational complexity is described in Table 2 of the Appendix, show that one may draw $\boldsymbol{y}$ from the unconstrained MVN as $\boldsymbol{y} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ and directly map it to a vector $\boldsymbol{x}$ on the intersection of hyperplanes using $\boldsymbol{x} = \boldsymbol{\Sigma}\mathbf{G}^T(\mathbf{G}\boldsymbol{\Sigma}\mathbf{G}^T)^{-1}\boldsymbol{r} + \left[\mathbf{I} - \boldsymbol{\Sigma}\mathbf{G}^T(\mathbf{G}\boldsymbol{\Sigma}\mathbf{G}^T)^{-1}\mathbf{G}\right]\boldsymbol{y}$. For illustration, with the same $\boldsymbol{\mu}$, $\boldsymbol{\Sigma}$, $\mathbf{G}$, and $\boldsymbol{r}$ as those in Figure 1, we show in Figure 2 a simple two dimensional example, where the unrestricted Gaussian distribution $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ is represented with a set of ellipses, and the constrained sample space $\mathcal{S}$ is represented as a straight line in the two-dimensional setting. With $\boldsymbol{\Sigma}\mathbf{G}^T(\mathbf{G}\boldsymbol{\Sigma}\mathbf{G}^T)^{-1}\boldsymbol{r} = (0.5, 0.5)^T$, $\left[\mathbf{I} - \boldsymbol{\Sigma}\mathbf{G}^T(\mathbf{G}\boldsymbol{\Sigma}\mathbf{G}^T)^{-1}\mathbf{G}\right] = [(0.5, -0.5)^T, (-0.5, 0.5)^T]$, one may directly maps a sample $\boldsymbol{y} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ to a vector on the constrained space. For example, if $\boldsymbol{y} = (1, 2)^T$, then it would be mapped to $\boldsymbol{x} = (0, 1)^T$ on the straight line.

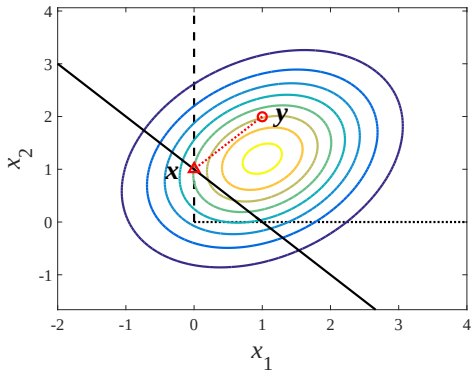



Figure 2: A two dimensional demonstration of Algorithm 2 that maps a random sample from $\boldsymbol{y} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ to a sample in the constrained space using $\boldsymbol{x} = \boldsymbol{\Sigma}\mathbf{G}^T(\mathbf{G}\boldsymbol{\Sigma}\mathbf{G}^T)^{-1}\boldsymbol{r} + \left[\mathbf{I} - \boldsymbol{\Sigma}\mathbf{G}^T(\mathbf{G}\boldsymbol{\Sigma}\mathbf{G}^T)^{-1}\mathbf{G}\right]\boldsymbol{y}$. For example, if $\boldsymbol{\mu} = (1, 1.2)^T$, $\boldsymbol{\Sigma} = [(1, 0.3)^T, (0.3, 1)^T]$, $\mathbf{G} = (1, 1)$, and $\boldsymbol{r} = 1$, then $\boldsymbol{y} = (1, 2)^T$ would be mapped to $\boldsymbol{x} = (0, 1)^T$ on a straight line using Algorithm 2.

## 3.1 Fast simulation of MVN distributions with structured covariance or precision matrices

For fast simulation of MVN distributions with structured covariance or precision matrices, our idea is to relate them to higher-dimensional hyperplane-truncated MVN distributions, with block-diagonal covariance matrices, that can be efficiently simulated with Algorithm 2. We first introduce an efficient algorithm for the simulation of a MVN distribution, whose covariance matrix is a positive-definite matrix subtracted by a low-rank symmetric matrix. Such kind of covariance matrices commonly arise in the conditional distributions of MVN distributions, as shown in (4). We then further extend this algorithm to the simulation of a MVN distribution whose precision (inverse covariance) matrix is the sum of a positive-definite matrix and a low-rank symmetric matrix. Such kind of precision matrices commonly arise in the conditional posterior distributions of the regression coefficients in both linear regression and generalized linear models.

**Theorem 2.** *The probability density function (PDF) of the MVN distribution*

$$\boldsymbol{x}_1 \sim \mathcal{N}(\boldsymbol{\mu}_1,\ \boldsymbol{\Sigma}_{11} - \boldsymbol{\Sigma}_{12}\boldsymbol{\Sigma}_{22}^{-1}\boldsymbol{\Sigma}_{21}), \tag{8}$$

*is the same as the PDF of the marginal distribution of* $\boldsymbol{x}_1 = (x_1, \ldots, x_{k_1})^T$ *in* $\boldsymbol{x} = (\boldsymbol{x}_1^T, x_{k_1+1}, \ldots, x_k)^T$*, whose PDF is expressed as*

$$\begin{aligned} p(\boldsymbol{x} \mid \boldsymbol{\mu}, \tilde{\boldsymbol{\Sigma}}, \mathbf{G}, \boldsymbol{r}) &= \mathcal{N}_{\{\boldsymbol{x}:\mathbf{G}\boldsymbol{x}=\boldsymbol{r}\}}\left(\boldsymbol{\mu}, \tilde{\boldsymbol{\Sigma}}\right) \\ &= \frac{1}{Z}\exp\left[-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^T\tilde{\boldsymbol{\Sigma}}^{-1}(\boldsymbol{x}-\boldsymbol{\mu})\right]\delta(\mathbf{G}\boldsymbol{x} = \boldsymbol{r}), \end{aligned} \tag{9}$$

*where $Z$ is a normalization constant, $\mathbf{G}_1 = \boldsymbol{\Sigma}_{21}\boldsymbol{\Sigma}_{11}^{-1}$ is a matrix of size $k_2 \times k_1$, $\mathbf{G}_2$ is a user-specified full rank invertible matrix of size $k_2 \times k_2$, $\boldsymbol{r} \in \mathbb{R}^{k_2}$ is a user-specified vector, and*

$$\mathbf{G} = (\mathbf{G}_1, \mathbf{G}_2) \in \mathbb{R}^{k_2 \times k}, \quad \boldsymbol{\mu} = \begin{bmatrix} \boldsymbol{\mu}_1 \\ \boldsymbol{\mu}_2 \end{bmatrix} \in \mathbb{R}^k, \quad \tilde{\boldsymbol{\Sigma}} = \begin{bmatrix} \boldsymbol{\Sigma}_{11} & \mathbf{0} \\ \mathbf{0} & \tilde{\boldsymbol{\Sigma}}_{22} \end{bmatrix} \in \mathbb{R}^{k \times k}, \tag{10}$$

*where*

$$\boldsymbol{\mu}_2 = \mathbf{G}_2^{-1}(\boldsymbol{r} - \boldsymbol{\Sigma}_{21}\boldsymbol{\Sigma}_{11}^{-1}\boldsymbol{\mu}_1), \tag{11}$$

$$\tilde{\boldsymbol{\Sigma}}_{22} = \mathbf{G}_2^{-1}\left(\boldsymbol{\Sigma}_{22} - \boldsymbol{\Sigma}_{21}\boldsymbol{\Sigma}_{11}^{-1}\boldsymbol{\Sigma}_{12}\right)(\mathbf{G}_2^{-1})^T. \tag{12}$$

The above theorem shows how the simulation of a MVN distribution, whose covariance matrix is a positive-definite matrix minus a symmetric matrix, can be realized by the simulation of a higher-dimensional hyperplane-truncated MVN distribution. By construction, it makes the covariance matrix $\tilde{\boldsymbol{\Sigma}}$ of the truncated-MVN be block diagonal, but still preserves the flexibility to customize the full-rank matrix $\mathbf{G}_2$ and the vector $\boldsymbol{r}$. While there are infinitely many choices for both $\mathbf{G}_2$ and $\boldsymbol{r}$, in the following discussion, we remove that flexibility by specifying $\mathbf{G}_2 = \mathbf{I}_{k_2}$, leading to $\mathbf{G} = (\mathbf{G}_1, \mathbf{G}_2) = (\boldsymbol{\Sigma}_{21}\boldsymbol{\Sigma}_{11}^{-1}, \mathbf{I}_{k_2})$, and $\boldsymbol{r} = \boldsymbol{\Sigma}_{21}\boldsymbol{\Sigma}_{11}^{-1}\boldsymbol{\mu}_1$. This specific setting of $\mathbf{G}_2$ and $\boldsymbol{r}$ leads to the following Corollary that is a special case of Theorem 2. Note that while we choose this specific setting in the paper, depending on the problems under study, other settings may lead to even more efficient simulation algorithms.

**Corollary 3.** *The PDF of the MVN distribution*

$$\boldsymbol{x}_1 \sim \mathcal{N}(\boldsymbol{\mu}_1, \boldsymbol{\Sigma}_{11} - \boldsymbol{\Sigma}_{12}\boldsymbol{\Sigma}_{22}^{-1}\boldsymbol{\Sigma}_{21}) \tag{13}$$

*is the same as the PDF of the marginal distribution of $\boldsymbol{x}_1$ in $\boldsymbol{x} = (\boldsymbol{x}_1^T, x_{k_1+1}, \ldots, x_k)^T$, whose PDF is expressed as*

$$\begin{aligned} p(\boldsymbol{x}) &= \mathcal{N}_{\{\boldsymbol{x}:\, \boldsymbol{\Sigma}_{21}\boldsymbol{\Sigma}_{11}^{-1}\boldsymbol{x}_1 + \boldsymbol{x}_2 = \boldsymbol{\Sigma}_{21}\boldsymbol{\Sigma}_{11}^{-1}\boldsymbol{\mu}_1\}}\left(\boldsymbol{\mu}, \tilde{\boldsymbol{\Sigma}}\right) \\ &= \frac{1}{Z}\exp\left[-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^T\tilde{\boldsymbol{\Sigma}}^{-1}(\boldsymbol{x}-\boldsymbol{\mu})\right]\delta(\boldsymbol{\Sigma}_{21}\boldsymbol{\Sigma}_{11}^{-1}\boldsymbol{x}_1 + \boldsymbol{x}_2 = \boldsymbol{\Sigma}_{21}\boldsymbol{\Sigma}_{11}^{-1}\boldsymbol{\mu}_1), \end{aligned} \tag{14}$$

*where $\boldsymbol{x}_2 = (x_{k_1+1}, \ldots, x_k)^T$, $Z$ is a normalization constant, and*

$$\boldsymbol{\mu} = \begin{bmatrix} \boldsymbol{\mu}_1 \\ \mathbf{0} \end{bmatrix} \in \mathbb{R}^k, \quad \tilde{\boldsymbol{\Sigma}} = \begin{bmatrix} \boldsymbol{\Sigma}_{11} & \mathbf{0} \\ \mathbf{0} & \boldsymbol{\Sigma}_{22} - \boldsymbol{\Sigma}_{21}\boldsymbol{\Sigma}_{11}^{-1}\boldsymbol{\Sigma}_{12} \end{bmatrix} \in \mathbb{R}^{k \times k}. \tag{15}$$

Further applying Theorem 1 to Corollary 3, as described in detail in the Appendix, a MVN random variable $\boldsymbol{x}$ with a structured covariance matrix can be generated as in Algorithm 3, where there is no need to compute $\boldsymbol{\Sigma}_{11} - \boldsymbol{\Sigma}_{12}\boldsymbol{\Sigma}_{22}^{-1}\boldsymbol{\Sigma}_{21}$ and its Cholesky decomposition. Suppose the covariance matrix $\boldsymbol{\Sigma}_{11}$ admits some special structure that makes it easy to invert and computationally efficient to simulate from $\mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma}_{11})$, then Algorithm 3 could lead to a significant saving in computation if $k_2 \ll k_1$. On the other

hand, when $k_2 \gg k_1$ and $\boldsymbol{\Sigma}_{22} - \boldsymbol{\Sigma}_{21}\boldsymbol{\Sigma}_{11}^{-1}\boldsymbol{\Sigma}_{12}$ admits no special structures, Algorithm 3 may not bring any computational advantage and hence one may resort to the naive Cholesky decomposition based procedure. Detailed computational complexity analyses for both methods are provided in Tables 3 and 4 of the Appendix, respectively.

---

**Algorithm 3** Simulation of the MVN distribution

$$\boldsymbol{x}_1 \sim \mathcal{N}(\boldsymbol{\mu}_1, \boldsymbol{\Sigma}_{11} - \boldsymbol{\Sigma}_{12}\boldsymbol{\Sigma}_{22}^{-1}\boldsymbol{\Sigma}_{21}).$$

---

- Sample $\boldsymbol{y}_1 \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{\Sigma}_{11})$ and $\boldsymbol{y}_2 \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{\Sigma}_{22} - \boldsymbol{\Sigma}_{21}\boldsymbol{\Sigma}_{11}^{-1}\boldsymbol{\Sigma}_{12})$ ;
- Return $\boldsymbol{x}_1 = \boldsymbol{\mu}_1 + \boldsymbol{y}_1 - \boldsymbol{\Sigma}_{12}\boldsymbol{\Sigma}_{22}^{-1}(\boldsymbol{\Sigma}_{21}\boldsymbol{\Sigma}_{11}^{-1}\boldsymbol{y}_1 + \boldsymbol{y}_2)$, which can be realized using
  - Solve $\boldsymbol{\alpha}$ such that $\boldsymbol{\Sigma}_{22}\boldsymbol{\alpha} = \boldsymbol{\Sigma}_{21}\boldsymbol{\Sigma}_{11}^{-1}\boldsymbol{y}_1 + \boldsymbol{y}_2$;
  - Return $\boldsymbol{x}_1 = \boldsymbol{\mu}_1 + \boldsymbol{y}_1 - \boldsymbol{\Sigma}_{12}\boldsymbol{\alpha}$.

---

**Corollary 4.** *A random variable simulated with Algorithm 3 is distributed as* $\boldsymbol{x}_1 \sim \mathcal{N}(\boldsymbol{\mu}_1, \boldsymbol{\Sigma}_{11} - \boldsymbol{\Sigma}_{12}\boldsymbol{\Sigma}_{22}^{-1}\boldsymbol{\Sigma}_{21})$.

The efficient simulation algorithm for a MVN distribution with a structured covariance matrix can also be further extended to a MVN distribution with a structured precision matrix, as described below, where $\boldsymbol{\beta} \in \mathbb{R}^p$, $\boldsymbol{\mu}_\beta \in \mathbb{R}^p$, $\boldsymbol{\Phi} \in \mathbb{R}^{n\times p}$, and both $\mathbf{A} \in \mathbb{R}^{p\times p}$ and $\boldsymbol{\Omega} \in \mathbb{R}^{n\times n}$ are positive-definite matrices. Computational complexity analyses for both the naive Cholesky decomposition based implementation and Algorithm 4 are provided in Table 5 and 6 of the Appendix, respectively. Similar to Algorithm 3, Algorithm 4 may bring a significant saving in computation when $p \gg n$ and $\mathbf{A}$ admits some special structure that makes it easy to invert and computationally efficient to simulate $\boldsymbol{y}_1$.

---

**Algorithm 4** Simulation of the MVN distribution

$$\boldsymbol{\beta} \sim \mathcal{N}\left[\boldsymbol{\mu}_\beta, (\mathbf{A} + \boldsymbol{\Phi}^T\boldsymbol{\Omega}\boldsymbol{\Phi})^{-1}\right].$$

---

- Sample $\boldsymbol{y}_1 \sim \mathcal{N}(\boldsymbol{0}, \mathbf{A}^{-1})$ and $\boldsymbol{y}_2 \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{\Omega}^{-1})$ ;
- Return $\boldsymbol{\beta} = \boldsymbol{\mu}_\beta + \boldsymbol{y}_1 - \mathbf{A}^{-1}\boldsymbol{\Phi}^T(\boldsymbol{\Omega}^{-1} + \boldsymbol{\Phi}\mathbf{A}^{-1}\boldsymbol{\Phi}^T)^{-1}(\boldsymbol{\Phi}\boldsymbol{y}_1 + \boldsymbol{y}_2)$, which can be realized using
  - Solve $\boldsymbol{\alpha}$ such that $(\boldsymbol{\Omega}^{-1} + \boldsymbol{\Phi}\mathbf{A}^{-1}\boldsymbol{\Phi}^T)\boldsymbol{\alpha} = \boldsymbol{\Phi}\boldsymbol{y}_1 + \boldsymbol{y}_2$.
  - Return $\boldsymbol{\beta} = \boldsymbol{\mu}_\beta + \boldsymbol{y}_1 - \mathbf{A}^{-1}\boldsymbol{\Phi}^T\boldsymbol{\alpha}$.

---

**Corollary 5.** *The random variable obtained with Algorithm 4 is distributed as* $\boldsymbol{\beta} \sim \mathcal{N}(\boldsymbol{\mu}_\beta, \boldsymbol{\Sigma}_\beta)$*, where* $\boldsymbol{\Sigma}_\beta = (\mathbf{A} + \boldsymbol{\Phi}^T \boldsymbol{\Omega} \boldsymbol{\Phi})^{-1}$.

# 4 Illustrations

Below we provide several examples to illustrate Theorem 1, which shows how to efficiently simulate from a hyperplane-truncated MVN distribution, and Corollary 4 (Corollary 5), which shows how to efficiently simulate from a MVN distribution with a structured covariance (precision) matrix. We run all our experiments on a 2.9 GHz computer.

## 4.1 Simulation of hyperplane-truncated MVNs

We first compare Algorithms 1 and 2, whose generated random samples follow the same distribution, as suggested by Theorem 1, to highlight the advantages of Algorithm 2 over Algorithm 1. We then employ Algorithm 2 for a real application whose data dimension is high and sample size is large.

### Comparison of Algorithms 1 and 2

We compare Algorithms 1 and 2 in a wide variety of settings by varying the data dimension $k$, varying the number of hyperplane constraints $k_2$, and choosing either a diagonal covariance matrix $\boldsymbol{\Sigma}$ or a non-diagonal one. We generate random diagonal covariance matrices using the MATLAB command $\text{diag}(0.05 + \text{rand}(k, 1))$ and random non-diagonal ones using $U.' * \text{diag}(0.05 + \text{rand}(k, 1)) * U$, where $\text{rand}(k, 1)$ is a vector of $k$ uniform random numbers and $U$ consists of a set of $k$ orthogonal basis vectors. The elements of $\boldsymbol{\mu}$, $\boldsymbol{r}$, and $\mathbf{G}$ are all sampled from $\mathcal{N}(0, 1)$, with the singular value decomposition applied to $\mathbf{G}$ to check whether $\text{Rank}(\mathbf{G}) = k_2$.

First, to verify Theorem 1, we conduct an experiment with $k = 5000$ data dimension, $k_2 = 20$ hyperplanes, and a diagonal $\boldsymbol{\Sigma}$. Contour plots of two randomly selected dimensions of the 10,000 random samples simulated with Algorithms 1 and 2 are shown in the top and bottom rows of Figure 3, respectively. The clear matches between the contour plots of these two different algorithms suggest the correctness of Theorem 1.

To demonstrate the efficiency of Algorithm 2, we first carry out a series of experiments with the number of hyperplane constraints fixed at $k_2 = 20$ and the data dimension increased from $k = 50$ to $k = 5000$. The computation time of simulating 10,000 samples averaged over five random trials is shown in Figure 4(a) for non-diagonal $\boldsymbol{\Sigma}$'s and in Figure 4(d) for diagonal ones. It is clear that, when the data dimension $k$ is high, Algorithm 2 has a clear advantage over Algorithm 1 by avoiding computing unnecessary intermediate variables, which is especially evident when $\boldsymbol{\Sigma}$ is diagonal. We then carry out a series of experiments where we vary not only $k$, but also $k_2$ from $0.1k$ to $0.9k$ for each $k$. As shown in Figure 4, it is evident that Algorithm 2 dominates Algorithm 1 in all scenarios, which can be explained by the fact that Algorithm 2 needs to compute much fewer intermediate variables. Also observed is that a larger $k_2$ leads to slower

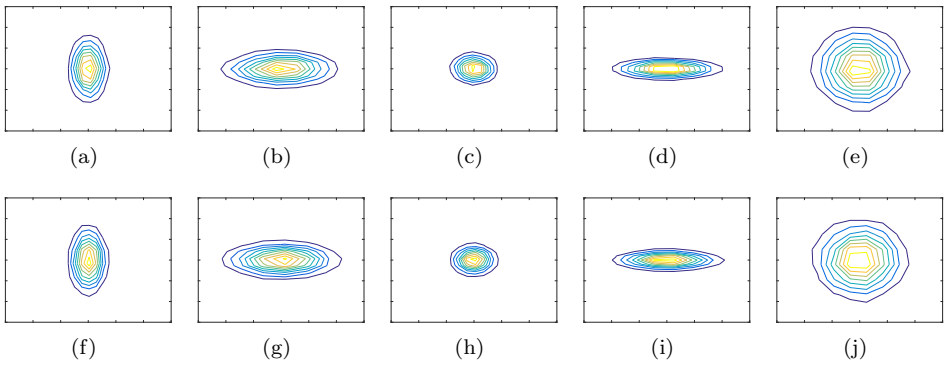



Figure 3: Comparison of the contour plots of two randomly selected dimensions of the 10,000 $k = 5000$ dimensional random samples simulated with Algorithm 1 (top row) and Algorithm 2 (bottom row). Each of the five columns corresponds to a random trial.

simulation for both algorithms, but to a much lesser extent for Algorithm 2. Moreover, the curvatures of those curves indicate that Algorithm 2 is more practical in a high dimensional setting. Note that since Algorithm 2 can naturally exploit the structure of the covariance matrix $\boldsymbol{\Sigma}$ for fast simulation, it is clearly more capable of benefiting from having a diagonal or block-diagonal $\boldsymbol{\Sigma}$, demonstrated by comparing Figures 4(b) and 4(c) with Figures 4(e) and 4(f). All these observations agree with our computational complexity analyses for Algorithms 1 and 2, as shown in Table 1 and 2 of the Appendix, respectively.

### A practical application of Algorithm 2

In what follows, we extend Algorithm 2 to facilitate simulation from a MVN distribution truncated on a probability simplex $\mathbb{S}^k = \{\boldsymbol{x} : \boldsymbol{x} \in \mathbb{R}^k, \mathbf{1}^T\boldsymbol{x} = 1, x_i \geq 0, i = 1, \cdots, k\}$. This problem frequently arises when unknown parameters can be interpreted as fractions or probabilities, for instance, in topic models (Blei et al. 2003), admixture models (Pritchard et al. 2000; Dobigeon et al. 2009b; Bazot et al. 2013), and discrete directed graphical models (Heckerman 1998). With Algorithm 2, one may remove the equality constraint to greatly simplify the problem.

More specifically, we focus on a big data setting in which the globally shared simplex-constrained model parameters could be linked to some latent counts via the multinomial likelihood. When there are tens of thousands or millions of observations in the dataset, scalable Bayesian inference for the simplex-constrained globally shared model parameters is highly desired, for example, for inferring the topics' distributions over words in latent Dirichlet allocation (Blei et al. 2003; Hoffman et al. 2010) and Poisson factor analysis (Zhou et al. 2012, 2016).

Let us denote the $\kappa$th model parameter vector constrained on a $V$-dimensional simplex by $\boldsymbol{\phi}_\kappa \in \mathbb{S}^V$, which could be linked to the latent counts $n_{vj\kappa} \in \mathbb{Z}$ of the $j$th

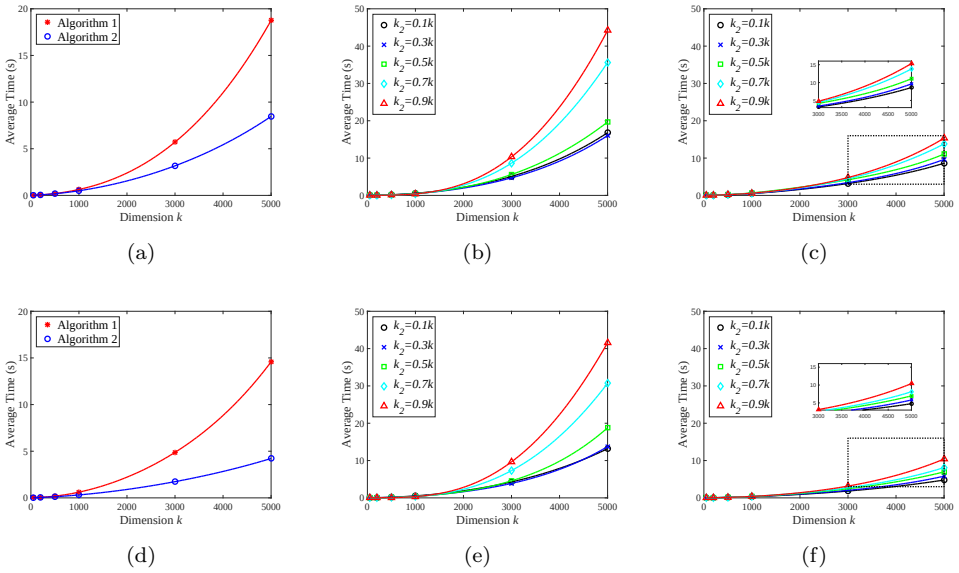



Figure 4: Average time of simulating 10,000 hyperplane-truncated MVN samples over five random trials in different dimensions with non-diagonal covariance matrixes (top row) and diagonal ones (bottom row). (a)(d) Comparison with fixed $k_2 = 20$. (b)(e) Algorithm 1 with varying $k_2$. (c)(f) Algorithm 2 with varying $k_2$.

document under a multinomial likelihood as $(n_{1j\kappa}, \ldots, n_{Vj\kappa}) \sim \text{Mult}(n_{\cdot j\kappa}, \boldsymbol{\phi}_\kappa)$, where $\mathbb{Z} = \{0, 1, 2, \cdots\}$, $v \in \{1, \ldots, V\}$, $\kappa \in \{1, \ldots, K\}$, and $j \in \{1, \ldots, N\}$. In topic modeling, one may consider $K$ as the total number of latent topics and $n_{vj\kappa}$ as the number of words at the $v$th vocabulary term in the $j$th document that are associated with the $\kappa$th latent topic. Note that the dimension $V$ in real applications is often large, such as tens of thousands in topic modeling. Given the observed counts $n_{vj}$ for the whole dataset, in a batch-learning setting, one typically iteratively updates the latent counts $n_{vj\kappa}$ conditioning on $\boldsymbol{\phi}_\kappa$, and updates $\boldsymbol{\phi}_\kappa$ conditioning on $n_{vj\kappa}$.

However, this batch-learning inference procedure would become inefficient and even impractical when the dataset size $N$ grows to a level that makes it too time consuming to finish even a single iteration of updating all local variables $n_{vj\kappa}$. To address this issue, we consider constructing a mini-batch based Bayesian inference procedure that could make substantial progress in posterior simulation while the batch-learning one may still be waiting to finish a single iteration.

Without loss of generality, in the following discussion, we drop the latent factor/topic index $\kappa$ to simplify the notation, focusing on the update of a single simplex-constrained global parameter vector. More specifically, we let the latent local count vector $\boldsymbol{n}_j = (n_{1j}, \ldots, n_{Vj})^T$ be linked to the simplex-constrained global parameter vector $\boldsymbol{\phi} \in \mathbb{S}^V$ via the multinomial likelihood as $\boldsymbol{n}_j \sim \text{Mult}(n_{\cdot j}, \boldsymbol{\phi})$, and impose a Dirichlet distribution

prior on $\boldsymbol{\phi}$ as $\boldsymbol{\phi} \sim \mathrm{Dir}\,(\eta \mathbf{1}_V)$.

Instead of waiting for all $\boldsymbol{n}_j$ to be updated before performing a single update of $\boldsymbol{\phi}$, we develop a mini-batch based Bayesian inference algorithm under a general framework for constructing stochastic gradient Markov chain Monte Carlo (SG-MCMC) (Ma et al. 2015), allowing $\boldsymbol{\phi}$ to be updated every time a mini-batch of $\boldsymbol{n}_j$ are processed. For the sake of completeness, we concisely describe the derivation for a SG-MCMC algorithm, as outlined below, for simplex-constrained globally shared model parameters. We refer the readers to Cong et al. (2017) for more details on the derivation and its application to scalable inference for topic modeling.

Using the reduced-mean parameterization of the simplex constrained vector $\boldsymbol{\phi}$, namely $\boldsymbol{\varphi} = (\phi_1, \cdots, \phi_{V-1})^T$, where $\boldsymbol{\varphi} \in \mathbb{R}_+^{V-1}$ is constrained with $\varphi_\cdot \leq 1$, we develop a SG-MCMC algorithm that updates $\boldsymbol{\varphi}$ for the $t$th mini-batch as

$$\boldsymbol{\varphi}_{t+1} = \left[\boldsymbol{\varphi}_t + \frac{\varepsilon_t}{M}[(\rho \bar{\boldsymbol{n}}_{:\cdot} + \eta) - (\rho n_{\cdot\cdot} + \eta V)\,\boldsymbol{\varphi}_t] + \mathcal{N}\left(\mathbf{0}, \frac{2\varepsilon_t}{M}\left[\mathrm{diag}\,(\boldsymbol{\varphi}_t) - \boldsymbol{\varphi}_t \boldsymbol{\varphi}_t^T\right]\right)\right]_{\triangle}, \quad (16)$$

where $\varepsilon_t$ are annealed step sizes, $\rho$ is the ratio of the dataset size $N$ to the mini-batch size, $\boldsymbol{n}_{:\cdot} = (\boldsymbol{n}_{1\cdot}, \cdots, \boldsymbol{n}_{V\cdot})^T = \sum_{j \in I_t} \boldsymbol{n}_j$, $\bar{\boldsymbol{n}}_{:\cdot} = (\boldsymbol{n}_{1\cdot}, \cdots, \boldsymbol{n}_{(V-1)\cdot})^T$, $[\cdot]_{\triangle}$ denotes the constraint that $\boldsymbol{\varphi} \in \mathbb{R}_+^{V-1}$ and $\varphi_\cdot \leq 1$, and $M := \mathbb{E}\left[\sum_{j=1}^N n_{\cdot j}\right]$ is approximated along the updating using $M = (1 - \varepsilon_t)\, M + \varepsilon_t \rho \mathbb{E}\,[n_{\cdot\cdot}]$. Alternatively, we have an equivalent update equation for $\boldsymbol{\phi}$ as

$$\boldsymbol{\phi}_{t+1} = \left[\boldsymbol{\phi}_t + \frac{\varepsilon_t}{M}[(\rho \boldsymbol{n}_{:\cdot} + \eta) - (\rho n_{\cdot\cdot} + \eta V)\,\boldsymbol{\phi}_t] + \mathcal{N}\left(\mathbf{0}, \frac{2\varepsilon_t}{M}\mathrm{diag}\,(\boldsymbol{\phi}_t)\right)\right]_{\angle}, \quad (17)$$

where $[\cdot]_{\angle}$ represents the constraint that $\boldsymbol{\phi} \in \mathbb{R}_+^V$ and $\mathbf{1}^T \boldsymbol{\phi} = 1$.

It is clear that (16) corresponds to simulation of a $V - 1$ dimensional truncated MVN distribution with $V$ inequality constraints. Since the number of constraints is larger than the dimension, previously proposed iterative simulation methods such as the one in Botev (2016) are often inappropriate. Note that, by omitting the non-negative constraints, the update in (17) corresponds to simulation of a hyperplane-truncated MVN simulation with a diagonal covariance matrix, which can be efficiently sampled as described in the following example.

**Example 1:** *Simulation of a hyperplane-truncated MVN distribution as*

$$\boldsymbol{x} \sim \mathcal{N}_{\mathcal{S}}[\boldsymbol{\mu}, a\,\mathrm{diag}(\boldsymbol{\phi})], \quad \mathcal{S} = \left\{\boldsymbol{x} : \mathbf{1}^T \boldsymbol{x} = 1\right\},$$

*where* $\boldsymbol{x} \in \mathbb{R}^k$, $\boldsymbol{\mu} \in \mathbb{R}^k$, $\mathbf{1}^T \boldsymbol{x} = \sum_{i=1}^k x_i$, $\boldsymbol{\phi} \in \mathbb{R}^k$, $a > 0$, $\phi_i > 0$ *for* $i \in \{1, \cdots, k\}$, *and* $\mathbf{1}^T \boldsymbol{\phi} = \sum_{i=1}^k \phi_i = 1$, *can be realized as follows.*

- *Sample* $\boldsymbol{y} \sim \mathcal{N}[\boldsymbol{\mu}, a\mathrm{diag}(\boldsymbol{\phi})]$*;*
- *Return* $\boldsymbol{x} = \boldsymbol{y} + (1 - \mathbf{1}^T \boldsymbol{y})\boldsymbol{\phi}$.

The sampling steps in Example 1 directly follow Algorithm 2 and Theorem 1 with the distribution parameters specified as $\mathbf{\Sigma} = a\text{diag}(\boldsymbol{\phi})$, $\mathbf{G} = \mathbf{1}^T$, and $\boldsymbol{r} = 1$. Accordingly, we present the following fast sampling procedure for (16).

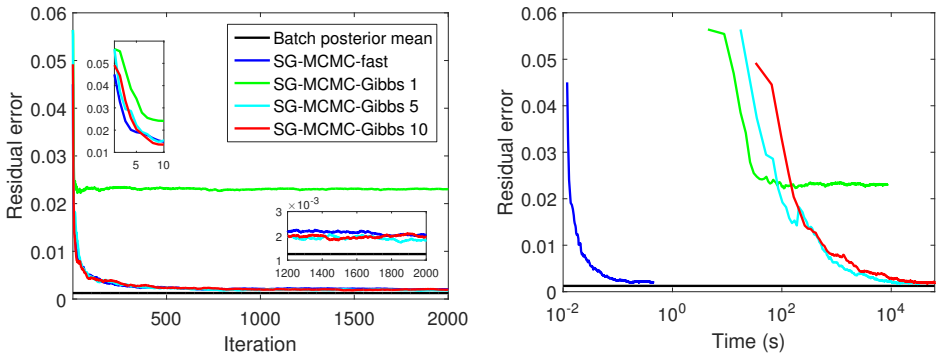
**Example 2:** *Simulation from* (16) *can be approximately but rapidly realized as*

- *Sample* $\boldsymbol{y} \sim \mathcal{N}\big[\boldsymbol{\phi}_t + \frac{\varepsilon_t}{M}\left[(\rho \boldsymbol{n}_{\cdot\cdot} + \eta) - (\rho n_{\cdot\cdot} + \eta V)\,\boldsymbol{\phi}_t\right], \frac{2\varepsilon_t}{M}\text{diag}\left(\boldsymbol{\phi}_t\right)\big]$*;*
- *Calculate* $\boldsymbol{z} = \boldsymbol{y} + (1 - \mathbf{1}^T\boldsymbol{y})\boldsymbol{\phi}_t$*;*
- *If* $\boldsymbol{z} \in \mathbb{S}$*, return* $\boldsymbol{\varphi}_{t+1} = (z_1, \cdots, z_{V-1})^T$*; else calculate* $\boldsymbol{d} = \max(\epsilon, \boldsymbol{z})$ *with a small constant* $\epsilon \geq 0$*, let* $\boldsymbol{e} = \boldsymbol{d}/\sum_{i=1}^{V} d_i$*, and return* $\boldsymbol{\varphi}_{t+1} = (e_1, \cdots, e_{V-1})^T$*.*

To verify Example 2, we conduct an experiment using multinomial-distributed data vectors of $V = 2000$ dimensions, which are generated as follows: considering that the simplex-constrained vector $\boldsymbol{\phi}$ is usually sparse in a high-dimensional application, we sample a $V = 2000$ dimensional vector $\boldsymbol{f}$ whose elements are uniformly distributed between 0 and 1, randomly select 40 dimensions and reset their values to be 100, and set $\boldsymbol{\phi} = \boldsymbol{f}/\sum_{i=1}^{V} f_i$; we simulate $N = 10,000$ samples, each $\boldsymbol{n}_j$ of which is generated from the multinomial distribution $\text{Mult}(n_{\cdot j}, \boldsymbol{\phi})$, where the number of trials is random and generated as $n_{\cdot j} \sim \text{Pois}(50)$. We set $\varepsilon_t = t^{-0.99}$ and use mini-batches, each of which consists of 10 data samples, to stochastically update global parameters via SG-MCMC.

For comparison, we choose the same SG-MCMC inference procedure but consider simulating (16), as performed every time a mini-batch of data samples are provided, either as in Example 2 or with the Gibbs sampler of Rodriguez-Yam et al. (2004). Simulating (16) with the Gibbs sampler of Rodriguez-Yam et al. (2004) is realized by updating all the $V$ dimensions, one dimension at a time, in each Gibbs sampling iteration. We set the total number of Gibbs sampling iterations for (16) in each mini-batch based update as 1, 5, or 10. Note that in practice, the $\boldsymbol{n}_j$ belonging to the current mini-batch are often latent and are updated conditioning on the data samples in the mini-batch and $\boldsymbol{\phi}$. For simplicity, all $\boldsymbol{n}_j$ here are simulated once and then fixed.

Using $\boldsymbol{\phi}^*_{post} = (\sum_{j=1}^{N} \boldsymbol{n}_j + \eta)/(\sum_{j=1}^{N} n_{\cdot j} + \eta V)$, the posterior mean of $\boldsymbol{\phi}$ in a batch-learning setting, as the reference, we show in Figure 5 how the residual errors for the estimated $\boldsymbol{\phi}^*$, defined as $\|\boldsymbol{\phi}^* - \boldsymbol{\phi}\|_2$, change both as a function of the number of processed mini-batches and as a function of computation time under various settings of the mini-batch based SG-MCMC algorithm. The curves shown in Figure 5 suggest that for each mini-batch, to simulate (16) with the Gibbs sampler of Rodriguez-Yam et al. (2004), it is necessary to have more than one Gibbs sampling iteration to achieve satisfactory results. It is clear from Figure 5(a) that the Gibbs sampler with 5 or 10 iterations for each mini-batch, even though each mini-batch has only 10 data samples, provides residual errors that quickly approach that of the batch posterior mean with a tiny gap, indicating the effectiveness of the SG-MCMC updating in (16). While simulating (16) with Gibbs sampling could in theory lead to unbiased samples if the number of Gibbs sampling iterations is large enough, it is much more efficient to simulate (16) with the procedure described in Example 2, which provides a performance that is undis-




(a) (b)

Figure 5: Comparisons of the residual errors of the simplex-constrained parameter vector, estimated under various settings of the stochastic-gradient MCMC (SG-MCMC) algorithm, as a function of (a) the number of processed mini-batches and (b) time. The curves labeled as "Batch posterior mean", "SG-MCMC-fast", and "SG-MCMC-Gibbs" correspond to the batch posterior mean, SG-MCMC with (16) simulated as in Example 2, and SG-MCMC with (16) simulated with the Gibbs sampler of Rodriguez-Yam et al. (2004), respectively. The digit following "SG-MCMC-Gibbs" represents the number of Gibbs sampling iterations to simulate (16) for each mini-batch.

tinguishable from those of the Gibbs sampler with as many as 5 or 10 iterations for each mini-batch, but at the expense of a tiny fraction of a single Gibbs sampling iteration.

## 4.2 Simulation of MVNs with structured covariance matrices

To illustrate Corollary 4, we mimic the truncated MVN simulation in (16) and present the following simulation example with a structured covariance matrix.

**Example 3:** Simulation of a MVN distribution as

$$\boldsymbol{x}_1 \sim \mathcal{N}[\boldsymbol{\mu}_1, a\,\mathrm{diag}(\boldsymbol{\phi}_1) - a\,\boldsymbol{\phi}_1\boldsymbol{\phi}_1^T],$$

where $\boldsymbol{x}_1 \in \mathbb{R}^{k-1}$, $\boldsymbol{\mu}_1 \in \mathbb{R}^{k-1}$, $a > 0$, $\boldsymbol{\phi}_1 = (\phi_1, \ldots, \phi_{k-1})^T$, $\phi_i > 0$ for $i \in \{1, \ldots, k-1\}$, and $\sum_{i=1}^{k-1} \phi_i < 1$, can be realized as follows.

- Sample $\boldsymbol{y}_1 \sim \mathcal{N}[\boldsymbol{0}, a\,\mathrm{diag}(\boldsymbol{\phi}_1)]$ and $\boldsymbol{y}_2 \sim \mathcal{N}(0, a^{-1}\,\phi_k)$, where $\phi_k = 1 - \sum_{i=1}^{k-1} \phi_i$;
- Return $\boldsymbol{x}_1 = \boldsymbol{\mu}_1 + \boldsymbol{y}_1 - (\boldsymbol{1}^T\boldsymbol{y}_1 + a\boldsymbol{y}_2)\boldsymbol{\phi}_1$.

Denoting $\boldsymbol{x} = (\boldsymbol{x}_1^T, x_k)^T$, $\boldsymbol{\phi} = (\boldsymbol{\phi}_1^T, \phi_k)^T$, $\boldsymbol{\mu} = (\boldsymbol{\mu}_1^T, \mu_k)^T$, and $\mu_k = 1 - \boldsymbol{1}^T\boldsymbol{\mu}_1$, the above sampling steps can also be equivalently expressed as follows.

- Sample $\boldsymbol{y} \sim \mathcal{N}[\boldsymbol{\mu}, a\,\mathrm{diag}(\boldsymbol{\phi})]$;

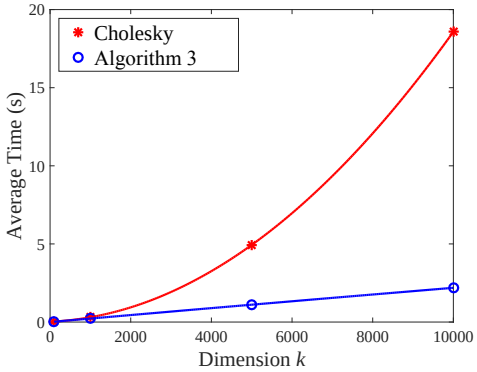



Figure 6: Comparison of the naive Cholesky decomposition based implementation and Algorithm 3 in terms of the average time of generating 10,000 $k$-dimensional random samples from $\boldsymbol{x}_1 \sim \mathcal{N}[\boldsymbol{\mu}_1, a\,\mathrm{diag}(\boldsymbol{\phi}_1) - a\,\boldsymbol{\phi}_1\boldsymbol{\phi}_1^T]$. The distribution parameters are randomly generated and computation time averaged over 100 random trials is displayed.

- Return $\boldsymbol{x}_1 = \boldsymbol{y}_1 + (1 - \mathbf{1}^T\boldsymbol{y})\boldsymbol{\phi}_1$.

Directly following Algorithm 3 and Corollary 4, the first sampling approach for the above example can be derived by specifying the distribution parameters as $\boldsymbol{\Sigma}_{11} = a\mathrm{diag}(\boldsymbol{\phi}_1)$, $\boldsymbol{\Sigma}_{12} = \boldsymbol{\phi}_1$, $\boldsymbol{\Sigma}_{21} = \boldsymbol{\phi}_1^T$, and $\boldsymbol{\Sigma}_{22} = a^{-1}$, while the second approach can be derived by specifying $\boldsymbol{\Sigma}_{11} = a\mathrm{diag}(\boldsymbol{\phi}_1)$, $\boldsymbol{\Sigma}_{12} = a\boldsymbol{\phi}_1$, $\boldsymbol{\Sigma}_{21} = a\boldsymbol{\phi}_1^T$, and $\boldsymbol{\Sigma}_{22} = a$.

To illustrate the efficiency of the proposed algorithms in Example 3, we simulate from the MVN distribution $\boldsymbol{x}_1 \sim \mathcal{N}[\boldsymbol{\mu}_1, a\,\mathrm{diag}(\boldsymbol{\phi}_1) - a\,\boldsymbol{\phi}_1\boldsymbol{\phi}_1^T]$ using both a naive implementation via Cholesky decomposition of the covariance matrix and the fast simulation algorithm for a hyperplane-truncated MVN random variable described in Example 3. We set the dimension from $k = 10^2$ up to $k = 10^4$ and set $\boldsymbol{\mu} = (1/k, \ldots, 1/k)$ and $a = 0.5$. For each $k$ and each simulation algorithm, we perform 100 independent random trials, in each of which $\boldsymbol{\phi}$ is sampled from the Dirichlet distribution $\mathrm{Dir}(1, \ldots, 1)$ and 10,000 independent random samples are simulated using that same $\boldsymbol{\phi}$.

As shown in Figure 6, for the proposed Algorithm 3, the average time of simulating 10,000 random samples increases linearly in the dimension $k$. By contrast, for the naive Cholesky decomposition based simulation algorithm, whose computational complexity is $O(k^3)$ (Golub and Van Loan 2012), the average simulation time increases at a significantly faster rate as the dimension $k$ increases.

For explicit verification, with the 10,000 simulated $k = 10^4$ dimensional random samples in a random trial, we randomly choose two dimensions and display their joint distribution using a contour plot. As in Figure 7, shown in the first row are the contour plots of five different random trials for the naive Cholesky implementation, whereas shown in the second row are the corresponding ones for the proposed Algorithm 3. As expected, the contour lines of the two figures in the same column closely match each other.

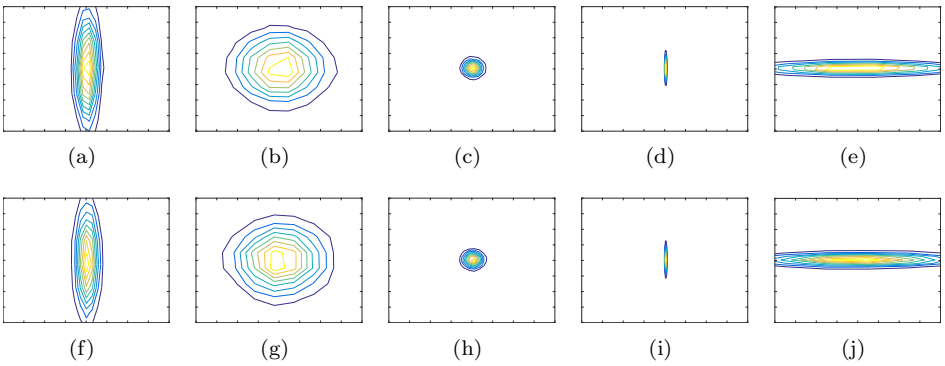



Figure 7: Comparison of the contour plots of two randomly selected dimensions of the 10,000 $k = 10^4$ dimensional random samples simulated with the naive Cholesky implementation (top row) and Algorithm 3 (bottom row). Each of the five columns corresponds to a random trial.

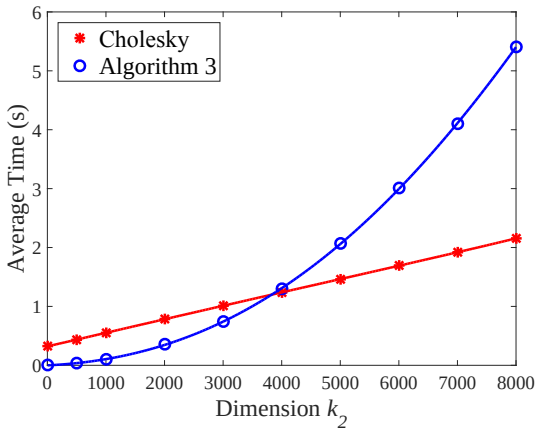



Figure 8: Comparison of the naive Cholesky decomposition based implementation and Algorithm 3 in terms of the average time of generating one $k_1 = 4000$ dimensional sample from $\boldsymbol{x}_1 \sim \mathcal{N}(\boldsymbol{\mu}_1, \boldsymbol{\Sigma}_{11} - \boldsymbol{\Sigma}_{12}\boldsymbol{\Sigma}_{22}^{-1}\boldsymbol{\Sigma}_{21})$, with diagonal $\boldsymbol{\Sigma}_{11}$ and $\boldsymbol{\Sigma}_{22}$. The distribution parameters are randomly generated and computation time averaged over 50 random trials is displayed.

To further examine when to apply Algorithm 3 instead of the naive Cholesky decomposition based implementation in a general setting, we present the computational complexity analyses in Tables 3 and 4 of the Appendix for the naive approach and Algorithm 3, respectively. In addition, we mimic the settings in Section 4.1 to conduct a set of experiments with randomly generated $\boldsymbol{\Sigma}_{12}$, diagonal $\boldsymbol{\Sigma}_{11}$, and diagonal $\boldsymbol{\Sigma}_{22}$. We fix $k_1 = 4000$ and vary $k_2$ from 1 to 8000. The computation time for one sample averaged over 50 random trials is presented in Figure 8. It is clear from Tables 3 and 4 and Figure 8 that, as a general guideline, one may choose Algorithm 3 when $k_2$ is

smaller than $k_1$ and $\mathbf{\Sigma}_{11}$ admits some special structure that makes it easy to invert and computationally efficient to simulate from $\mathcal{N}(\mathbf{0}, \mathbf{\Sigma_{11}})$.

## 4.3 Simulation of MVNs with structured precision matrices

To examine when to apply Algorithm 4 instead of the naive Choleskey decomposition based procedure, we first consider a series of random simulations in which the sample size $n$ is fixed while the data dimension $p$ is varying. We then show that Algorithm 4 can be applied for high-dimensional regression whose $p$ is often much larger than $n$.

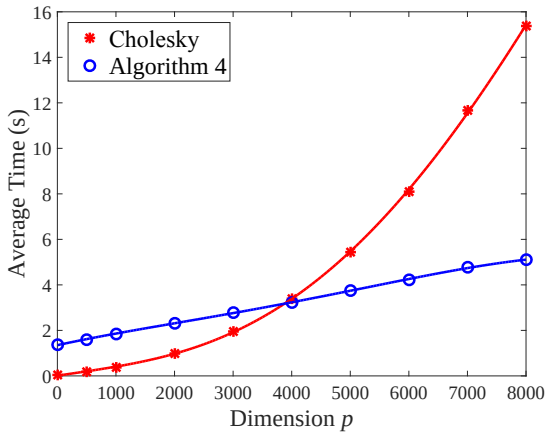



Figure 9: Comparison of the naive Cholesky decomposition based implementation and Algorithm 4 in terms of the average time of generating one $p$ dimensional sample from $\boldsymbol{\beta} \sim \mathcal{N}\left[\boldsymbol{\mu}_\beta, (\mathbf{A} + \mathbf{\Phi}^T \mathbf{\Omega} \mathbf{\Phi})^{-1}\right]$, with diagonal $\mathbf{A}$ and $\mathbf{\Omega}$. The distribution parameters are randomly generated and computation time averaged over 50 random trials is displayed.

We fix $n = 4000$, vary $p$ from 1 to 8000, and mimic the settings in Section 4.1 to randomly generate $\mathbf{\Phi}$, diagonal $\mathbf{A}$, and diagonal $\mathbf{\Omega}$. As a function of dimensions $p$, the computation time for one sample averaged over 50 random trials is shown in Figure 9. It is evident that, identical to the complexity analysis in Tables 5 and 6, Algorithm 4 has a linear complexity with respect to $p$ under these settings, which will bring significant acceleration in a high-dimensional setting with $p \gg n$. If the sample size $n$ is large enough that $n > p$, then one may directly apply the naive Cholesky decomposition based implementation.

Algorithm 4 could be slightly modified to be applied to high-dimensional regression, where the main objective is to efficiently sample from the conditional posterior of $\boldsymbol{\beta} \in \mathbb{R}^{p \times 1}$ in the linear regression model as

$$\boldsymbol{t} \sim \mathcal{N}(\mathbf{\Phi}\boldsymbol{\beta}, \mathbf{\Omega}^{-1}), \ \boldsymbol{\beta} \sim \mathcal{N}(\mathbf{0}, \mathbf{A}^{-1}), \tag{18}$$

where $\mathbf{\Phi} \in \mathbb{R}^{n \times p}$, $\mathbf{\Omega} \in \mathbb{R}^{n \times n}$, and different constructions on $\mathbf{A} \in \mathbb{R}^{p \times p}$ lead to a wide variety of regression models (Caron and Doucet 2008; Carvalho et al. 2010; Polson et al.

2014). The conditional posterior of $\boldsymbol{\beta}$ is directly derived and shown in the following example, where its simulation algorithm is summarized by further generalizing Corollary 5.

**Example 4:** *Simulation of the MVN distribution*

$$\boldsymbol{\beta} \sim \mathcal{N}\left[(\mathbf{A}+\boldsymbol{\Phi}^T\boldsymbol{\Omega}\boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}^T\boldsymbol{\Omega}\boldsymbol{t},\ \ (\mathbf{A}+\boldsymbol{\Phi}^T\boldsymbol{\Omega}\boldsymbol{\Phi})^{-1}\right]$$

*can be realized as follows.*

- *Sample* $\boldsymbol{y}_1 \sim \mathcal{N}(\mathbf{0}, \mathbf{A}^{-1})$ *and* $\boldsymbol{y}_2 \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Omega}^{-1})$ *;*
- *Return* $\boldsymbol{\beta} = \boldsymbol{y}_1 + \mathbf{A}^{-1}\boldsymbol{\Phi}^T(\boldsymbol{\Omega}^{-1}+\boldsymbol{\Phi}\mathbf{A}^{-1}\boldsymbol{\Phi}^T)^{-1}(\boldsymbol{t}-\boldsymbol{\Phi}\boldsymbol{y}_1-\boldsymbol{y}_2)$*, which can be realized using*
  - *Solve* $\boldsymbol{\alpha}$ *such that* $(\boldsymbol{\Omega}^{-1}+\boldsymbol{\Phi}\mathbf{A}^{-1}\boldsymbol{\Phi}^T)\boldsymbol{\alpha} = \boldsymbol{t}-\boldsymbol{\Phi}\boldsymbol{y}_1-\boldsymbol{y}_2$*;*
  - *Return* $\boldsymbol{\beta} = \boldsymbol{y}_1 + \mathbf{A}^{-1}\boldsymbol{\Phi}^T\boldsymbol{\alpha}$*.*

Note that if $\boldsymbol{\Omega} = \mathbf{I}_n$, then the simulation algorithm in Example 4 reduces to the one in Proposition 2.1 of Bhattacharya et al. (2016), which is shown there to be significantly more efficient than that of Rue (2001) for high-dimensional regression if $p \gg n$.

# 5 Conclusions

A fast and exact simulation algorithm is developed for a multivariate normal (MVN) distribution whose sample space is constrained on the intersection of a set of hyperplanes, which is shown to be inherently related to the conditional distribution of a unconstrained MVN distribution. The proposed simulation algorithm is further generalized to efficiently simulate from a MVN distribution, whose covariance (precision) matrix can be decomposed as the sum (difference) of a positive-definite matrix and a low-rank symmetric matrix, using a higher dimensional hyperplane-truncated MVN distribution whose covariance matrix is block-diagonal.

# References

Albert, J. H. and Chib, S. (1993). "Bayesian analysis of binary and polychotomous response data." *J. Amer. Statist. Assoc.*, 88(422): 669–679.

Altmann, Y., McLaughlin, S., and Dobigeon, N. (2014). "Sampling from a multivariate Gaussian distribution truncated on a simplex: a review." In *Statistical Signal Processing (SSP), 2014 IEEE Workshop on*, 113–116. IEEE.

Bazot, C., Dobigeon, N., Tourneret, J.-Y., Zaas, A. K., Ginsburg, G. S., and Hero III, A. O. (2013). "Unsupervised Bayesian linear unmixing of gene expression microarrays." *BMC Bioinformatics*, 14(1): 1.

Bhattacharya, A., Chakraborty, A., and Mallick, B. K. (2016). "Fast sampling with Gaussian scale mixture priors in high-dimensional regression." *Biometrika*, 103(4): 985.

Blei, D. M., Ng, A. Y., and Jordan, M. I. (2003). "Latent Dirichlet allocation." *JMLR*, 3: 993–1022.

Botev, Z. (2016). "The normal law under linear restrictions: simulation and estimation via minimax tilting." *J. Roy. Statist. Soc.: Series B*.

Caron, F. and Doucet, A. (2008). "Sparse Bayesian nonparametric regression." In *ICML*, 88–95. ACM.

Carvalho, C. M., Polson, N. G., and Scott, J. G. (2010). "The horseshoe estimator for sparse signals." *Biometrika*, 97(2): 465–480.

Chopin, N. (2011). "Fast simulation of truncated Gaussian distributions." *Statistics and Computing*, 21(2): 275–288.

Cong, Y., Chen, B., and Zhou, M. (2017). "Topic-layer-adaptive stochastic gradient Riemannian (TLASGR) MCMC for deep latent Dirichlet allocation." *Preprint*.

Damien, P. and Walker, S. G. (2001). "Sampling truncated normal, beta, and gamma densities." *Journal of Computational and Graphical Statistics*, 10(2): 206–215.

Dobigeon, N., Moussaoui, S., Coulon, M., Tourneret, J.-Y., and Hero, A. O. (2009a). "Joint Bayesian endmember extraction and linear unmixing for hyperspectral imagery." *IEEE Transactions on Signal Processing*, 57(11): 4355–4368.

Dobigeon, N., Moussaoui, S., Tourneret, J.-Y., and Carteret, C. (2009b). "Bayesian separation of spectral sources under non-negativity and full additivity constraints." *Signal Processing*, 89(12): 2657–2669.

Doucet, A. (2010). "A note on efficient conditional simulation of Gaussian distributions." *Departments of Computer Science and Statistics, University of British Columbia*.

Gelfand, A. E., Smith, A. F., and Lee, T.-M. (1992). "Bayesian analysis of constrained parameter and truncated data problems using Gibbs sampling." *J. Amer. Statist. Assoc.*, 87(418): 523–532.

Gelfand, A. E. and Smith, A. F. M. (1990). "Sampling-based approaches to calculating marginal densities." *J. Amer. Statist. Assoc.*, 85(410): 398–409.

Geman, S. and Geman, D. (1984). "Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images." *IEEE Trans. Pattern Anal. Mach. Intell.*, 721–741.

Geweke, J. (1991). "Efficient simulation from the multivariate normal and student-t distributions subject to linear constraints and the evaluation of constraint probabilities." In *Computing science and statistics: Proceedings of the 23rd symposium on the interface*, 571–578.

Geweke, J. F. (1996). "Bayesian inference for linear models subject to linear inequality constraints." In *Modelling and Prediction Honoring Seymour Geisser*, 248–263. Springer.

Girolami, M. and Calderhead, B. (2011). "Riemann manifold Langevin and Hamiltonian Monte Carlo methods." *J. Roy. Statist. Soc.: B*, 73(2): 123–214.

Golub, G. H. and Van Loan, C. F. (2012). *Matrix computations*, volume 3. JHU Press.

Heckerman, D. (1998). "A tutorial on learning with Bayesian networks." In *Learning in graphical models*, 301–354. Springer.

Hoffman, M., Blei, D., and Bach, F. (2010). "Online learning for latent Dirichlet allocation." In *NIPS*.

Hoffman, Y. and Ribak, E. (1991). "Constrained realizations of Gaussian fields-A simple algorithm." *The Astrophysical Journal*, 380: L5–L8.

Holmes, C. C. and Held, L. (2006). "Bayesian auxiliary variable models for binary and multinomial regression." *Bayesian Analysis*, 1(1): 145–168.

Imai, K. and van Dyk, D. A. (2005). "A Bayesian analysis of the multinomial probit model using marginal data augmentation." *Journal of Econometrics*, 124(2): 311–334.

Johndrow, J., Dunson, D., and Lum, K. (2013). "Diagonal orthant multinomial probit models." In *AISTATS*, 29–38.

Lan, S., Zhou, B., and Shahbaba, B. (2014). "Spherical Hamiltonian Monte Carlo for constrained target distributions." In *ICML*, 629–637.

Ma, Y., Chen, T., and Fox, E. (2015). "A complete recipe for stochastic gradient MCMC." In *NIPS*, 2899–2907.

McCulloch, R. E., Polson, N. G., and Rossi, P. E. (2000). "A Bayesian analysis of the multinomial probit model with fully identified parameters." *Journal of Econometrics*, 99(1): 173–193.

Neelon, B. and Dunson, D. B. (2004). "Bayesian isotonic regression and trend analysis." *Biometrics*, 60(2): 398–406.

Pakman, A. and Paninski, L. (2014). "Exact Hamiltonian Monte Carlo for truncated multivariate Gaussians." *Journal of Computational and Graphical Statistics*, 23(2): 518–542.

Patterson, S. and Teh, Y. W. (2013). "Stochastic gradient Riemannian Langevin dynamics on the probability simplex." In *NIPS*, 3102–3110.

Polson, N. G., Scott, J. G., and Windle, J. (2014). "The Bayesian bridge." *J. Roy. Statist. Soc.: Series B*, 76(4): 713–733.

Pritchard, J. K., Stephens, M., and Donnelly, P. (2000). "Inference of population structure using multilocus genotype data." *Genetics*, 155(2): 945–959.

Robert, C. P. (1995). "Simulation of truncated normal variables." *Statistics and Computing*, 5(2): 121–125.

Rodriguez-Yam, G., Davis, R. A., and Scharf, L. L. (2004). "Efficient Gibbs sampling of truncated multivariate normal with application to constrained linear regression." *Technical report*.

Rue, H. (2001). "Fast sampling of Gaussian Markov random fields." *J. Roy. Statist. Soc.: Series B*, 63(2): 325–338.

Schmidt, M. (2009). "Linearly constrained Bayesian matrix factorization for blind source separation." In *NIPS*, 1624–1632.

Tong, Y. L. (2012). *The multivariate normal distribution*. Springer Science & Business Media.

Train, K. E. (2009). *Discrete choice methods with simulation*. Cambridge University Press.

Zhou, M., Cong, Y., and Chen, B. (2016). "Augmentable Gamma Belief Networks." *J. Mach. Learn. Res.*, 17(163): 1–44.

Zhou, M., Hannah, L., Dunson, D. B., and Carin, L. (2012). "Beta-negative binomial process and Poisson factor analysis." In *AISTATS*, 1462–1471.

**Acknowledgments**

The authors would like to thank the editor-in-chief, editor, associate editor, and two anonymous referees for their comments and suggestions, which have helped us improve the paper substantially. M. Zhou thanks Yingbo Li and Xiaojing Wang for helpful discussions. B. Chen thanks the support of the Thousand Young Talent Program of China, NSFC (61372132), NCET-13-0945, and NDPR-9140A07010115DZ01015.

# Appendix

**Algorithm 5** (Hoffman and Ribak 1991; Doucet 2010) Simulation of the conditional distribution of $\boldsymbol{x}_1$ given $\boldsymbol{x}_2 = \boldsymbol{r}$ as $\boldsymbol{x}_1 \,|\, \boldsymbol{x}_2 = \boldsymbol{r} \sim \mathcal{N}\left[\boldsymbol{\mu}_1 + \boldsymbol{\Sigma}_{12}\boldsymbol{\Sigma}_{22}^{-1}(\boldsymbol{r} - \boldsymbol{\mu}_2),\ \boldsymbol{\Sigma}_{11} - \boldsymbol{\Sigma}_{12}\boldsymbol{\Sigma}_{22}^{-1}\boldsymbol{\Sigma}_{21}\right]$, where the joint distribution of $\boldsymbol{x} = (\boldsymbol{x}_1^T, \boldsymbol{x}_2^T)$ follows $\boldsymbol{x} \sim (\boldsymbol{\mu}, \boldsymbol{\Sigma})$.

- Sample $\boldsymbol{y} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ and denote $\boldsymbol{y}_1 = (y_1, \ldots, y_{k_1})^T$ and $\boldsymbol{y}_2 = (y_{k_1+1}, \ldots, y_k)^T$;
- Return $\boldsymbol{x}_1 = \boldsymbol{y}_1 + \boldsymbol{\Sigma}_{12}\boldsymbol{\Sigma}_{22}^{-1}(\boldsymbol{r} - \boldsymbol{y}_2)$.

## Proofs

*Proof of Theorem 1.* Let us denote $\boldsymbol{\Lambda} = \mathbf{H}^T\boldsymbol{\Sigma}^{-1}\mathbf{H}$ as a precision matrix that can be partitioned as in (7). For Algorithm 1, instead of directly simulating $\boldsymbol{z}_1$ given $\boldsymbol{z}_2$ using the conditional distribution of the MVN, we apply Algorithm 5 (Hoffman and Ribak 1991; Doucet 2010) to modify its sampling steps as follows.

- Sample $\tilde{\boldsymbol{z}} \sim \mathcal{N}[\mathbf{H}^{-1}\boldsymbol{\mu}, \mathbf{H}^{-1}\boldsymbol{\Sigma}(\mathbf{H}^{-1})^T]$, and denote $\tilde{\boldsymbol{z}}_1 = (z_1, \ldots, z_{k_1})^T$ and $\tilde{\boldsymbol{z}}_2 = (z_{k_1+1}, \ldots, z_k)$;
- Let $\boldsymbol{z} = (\boldsymbol{z}_1^T, \boldsymbol{z}_2^T)^T$, where $\boldsymbol{z}_2 = (\mathbf{G}\mathbf{H}_2)^{-1}\boldsymbol{r}$ and $\boldsymbol{z}_1 = \tilde{\boldsymbol{z}}_1 - \boldsymbol{\Lambda}_{11}^{-1}\boldsymbol{\Lambda}_{12}(\boldsymbol{z}_2 - \tilde{\boldsymbol{z}}_2)$, and return

$$\begin{aligned}\boldsymbol{x} = \mathbf{H}\boldsymbol{z} &= \mathbf{H}_1\boldsymbol{z}_1 + \mathbf{H}_2(\mathbf{G}\mathbf{H}_2)^{-1}\boldsymbol{r} \\ &= \mathbf{H}_1\tilde{\boldsymbol{z}}_1 + \mathbf{H}_1\boldsymbol{\Lambda}_{11}^{-1}\boldsymbol{\Lambda}_{12}\tilde{\boldsymbol{z}}_2 + (\mathbf{H}_2 - \mathbf{H}_1\boldsymbol{\Lambda}_{11}^{-1}\boldsymbol{\Lambda}_{12})(\mathbf{G}\mathbf{H}_2)^{-1}\boldsymbol{r} \\ &= (\mathbf{H}_1, \mathbf{H}_1\boldsymbol{\Lambda}_{11}^{-1}\boldsymbol{\Lambda}_{12})\tilde{\boldsymbol{z}} + (\mathbf{H}_2 - \mathbf{H}_1\boldsymbol{\Lambda}_{11}^{-1}\boldsymbol{\Lambda}_{12})(\mathbf{G}\mathbf{H}_2)^{-1}\boldsymbol{r}. \end{aligned} \tag{19}$$

Therefore, we can equivalently generate $\boldsymbol{x}$ as follows.

- Sample $\boldsymbol{y} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$.
- Return $\boldsymbol{x} = (\mathbf{H}_1, \mathbf{H}_1\boldsymbol{\Lambda}_{11}^{-1}\boldsymbol{\Lambda}_{12})\mathbf{H}^{-1}\boldsymbol{y} + (\mathbf{H}_2 - \mathbf{H}_1\boldsymbol{\Lambda}_{11}^{-1}\boldsymbol{\Lambda}_{12})(\mathbf{G}\mathbf{H}_2)^{-1}\boldsymbol{r}$.

The computation can be significantly simplified if $\boldsymbol{\Lambda}_{12} = \mathbf{0}$, which means

$$\boldsymbol{\Lambda}_{12} = \mathbf{H}_1^T\boldsymbol{\Sigma}^{-1}\mathbf{H}_2 = \mathbf{0}.$$

Since $\mathbf{H}_1^T\mathbf{G}^T = 0$ by definition, to make $\boldsymbol{\Lambda}_{12} = \mathbf{0}$, if and only if we have $\mathbf{H}_2$ as

$$\mathbf{H}_2 = \boldsymbol{\Sigma}\mathbf{G}^T\mathbf{M},$$

where $\mathbf{M} \in \mathbb{R}^{k_2 \times k_2}$ is an arbitrary full rank matrix, under which we have

- Sample $\boldsymbol{y} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$.
- Return $\boldsymbol{x} = (\mathbf{H}_1, \mathbf{0}_{k\times k_2})\mathbf{H}^{-1}\boldsymbol{y} + \boldsymbol{\Sigma}\mathbf{G}^T(\mathbf{G}\boldsymbol{\Sigma}\mathbf{G}^T)^{-1}\boldsymbol{r}$, or return

$$\boldsymbol{x} = \boldsymbol{y} - (\mathbf{0}_{k\times k_1}, \boldsymbol{\Sigma}\mathbf{G}^T\mathbf{M})\mathbf{H}^{-1}\boldsymbol{y} + \boldsymbol{\Sigma}\mathbf{G}^T(\mathbf{G}\boldsymbol{\Sigma}\mathbf{G}^T)^{-1}\boldsymbol{r}. \tag{20}$$

Let us denote $(\mathbf{0}_{k\times k_1}, \boldsymbol{\Sigma}\mathbf{G}^T\mathbf{M})\mathbf{H}^{-1} = \mathbf{C}$. We have

$$(\mathbf{0}_{k\times k_1}, \boldsymbol{\Sigma}\mathbf{G}^T\mathbf{M}) = \mathbf{C}\mathbf{H} = (\mathbf{C}\mathbf{H}_1, \mathbf{C}\boldsymbol{\Sigma}\mathbf{G}^T\mathbf{M})$$

and hence $\mathbf{C}\mathbf{H}_1 = \mathbf{0}$ and $\mathbf{C}\boldsymbol{\Sigma}\mathbf{G}^T\mathbf{M} = \boldsymbol{\Sigma}\mathbf{G}^T\mathbf{M}$. Since $\mathbf{G}\mathbf{H}_1 = 0$, we have $\mathbf{C} = \boldsymbol{\Sigma}\mathbf{G}^T(\mathbf{G}\boldsymbol{\Sigma}\mathbf{G}^T)^{-1}\mathbf{G}$. The proof is completed by substituting $(\mathbf{0}_{k\times k_1}, \boldsymbol{\Sigma}\mathbf{G}^T\mathbf{M})\mathbf{H}^{-1}$ in (20) with $\boldsymbol{\Sigma}\mathbf{G}^T(\mathbf{G}\boldsymbol{\Sigma}\mathbf{G}^T)^{-1}\mathbf{G}$. □

*Alternative Proof of Theorem 1.* To solve the problem in (3), one may solve an equivalent problem in (6) by defining an invertible transformation matrix $\mathbf{H}$ that satisfies $\mathbf{G}\mathbf{H}_1 = \mathbf{0}_{k_2\times k_1}$. Let us denote $\boldsymbol{\Lambda} = \mathbf{H}^T\boldsymbol{\Sigma}^{-1}\mathbf{H}$ as a precision matrix that can be partitioned as in (7). To simply the problem in (6), we choose the transformation matrix $\mathbf{H}$ to make $\boldsymbol{z}_1$ and $\boldsymbol{z}_2$ be independent to each other. Since $\boldsymbol{z}$ follows a MVN distribution, $\boldsymbol{z}_1$ and $\boldsymbol{z}_2$ are independent to each other if and only if

$$\boldsymbol{\Lambda}_{12} = \mathbf{H}_1^T\boldsymbol{\Sigma}^{-1}\mathbf{H}_2 = \mathbf{0}.$$

Since $\mathbf{H}_1^T\mathbf{G}^T = \mathbf{0}$ by definition, to make $\boldsymbol{\Lambda}_{12} = \mathbf{0}$, if and only if we have $\mathbf{H}_2$ as

$$\mathbf{H}_2 = \boldsymbol{\Sigma}\mathbf{G}^T\mathbf{M},$$

where $\mathbf{M} \in \mathbb{R}^{k_2\times k_2}$ is an arbitrary full rank matrix. Accordingly, we have

$$\mathbf{H}^{-1}\boldsymbol{\Sigma}(\mathbf{H}^{-1})^T = \begin{bmatrix} (\mathbf{H}_1^T\boldsymbol{\Sigma}^{-1}\mathbf{H}_1)^{-1} & \mathbf{0} \\ \mathbf{0} & (\mathbf{H}_2^T\boldsymbol{\Sigma}^{-1}\mathbf{H}_2)^{-1} \end{bmatrix}.$$

Thus with $\mathbf{H}$ satisfying $\mathbf{G}\mathbf{H}_1 = \mathbf{0}_{k_2\times k_1}$ and $\mathbf{H}_2 = \boldsymbol{\Sigma}\mathbf{G}^T\mathbf{M}$, one can transform the original problem in (3) to that in (6), where $\boldsymbol{z}_1$ and $\boldsymbol{z}_2$ are independent and the restrictions $\mathbf{G}\boldsymbol{x} = \boldsymbol{r}$ and $\boldsymbol{z}_2 = (\mathbf{G}\mathbf{H}_2)^{-1}\boldsymbol{r}$ imply each other. Following the naive approach shown in Algorithm 1, one can generate $\boldsymbol{x}$ from (3) as follows

- Find $\mathbf{H} = (\mathbf{H}_1, \mathbf{H}_2)$ with $\mathbf{H}_2 = \boldsymbol{\Sigma}\mathbf{G}^T\mathbf{M}$ and with $\mathbf{H}_1$ satisfying $\mathbf{G}\mathbf{H}_1 = \mathbf{0}_{k_2\times k_1}$;
- Sample $\boldsymbol{z}_1 \sim \mathcal{N}[(\mathbf{I}_{k_1}, \mathbf{0}_{k_1\times k_2})\mathbf{H}^{-1}\boldsymbol{\mu}, (\mathbf{H}_1^T\boldsymbol{\Sigma}^{-1}\mathbf{H}_1)^{-1}]$;
- Return $\boldsymbol{x} = \mathbf{H}\begin{bmatrix} \boldsymbol{z}_1 \\ \mathbf{M}^{-1}(\mathbf{G}\boldsymbol{\Sigma}\mathbf{G}^T)^{-1}\boldsymbol{r} \end{bmatrix}$.

However, this naive approach contains intermediate variables that could be computationally expensive to compute. Below we present a method to bypass these intermediate variables. Since the last step could be reexpressed as

$$\begin{aligned} \boldsymbol{x} &= \mathbf{H}_1\boldsymbol{z}_1 + \mathbf{H}_2\mathbf{M}^{-1}(\mathbf{G}\boldsymbol{\Sigma}\mathbf{G}^T)^{-1}\boldsymbol{r} \\ &= \mathbf{H}_1\boldsymbol{z}_1 + \mathbf{H}_2\boldsymbol{z}_2 + \mathbf{H}_2[\mathbf{M}^{-1}(\mathbf{G}\boldsymbol{\Sigma}\mathbf{G}^T)^{-1}\boldsymbol{r} - \boldsymbol{z}_2] \\ &= \mathbf{H}\boldsymbol{z} + \boldsymbol{\Sigma}\mathbf{G}^T(\mathbf{G}\boldsymbol{\Sigma}\mathbf{G}^T)^{-1}\boldsymbol{r} - \boldsymbol{\Sigma}\mathbf{G}^T(\mathbf{M}\boldsymbol{z}_2), \end{aligned}$$

where $\boldsymbol{z} = (\boldsymbol{z}_1^T, \boldsymbol{z}_2^T)^T$ and $\boldsymbol{z}_2 \in \mathbb{R}^{k_2}$ is a vector whose values can be chosen arbitrarily. In addition, since $\mathbf{M}$ is an arbitrary full-rank matrix, we can let

$$\boldsymbol{z}_2 \sim \mathcal{N}[(\mathbf{0}_{k_2 \times k_1}, \mathbf{I}_{k_2})\mathbf{H}^{-1}\boldsymbol{\mu}, (\mathbf{H}_2^T \boldsymbol{\Sigma}^{-1} \mathbf{H}_2)^{-1}],$$

which means $\boldsymbol{z} \sim \mathcal{N}[\mathbf{H}^{-1}\boldsymbol{\mu}, \mathbf{H}^{-1}\boldsymbol{\Sigma}(\mathbf{H}^{-1})^T]$, and choose $\mathbf{M}$ to make

$$\mathbf{G}\boldsymbol{x} = \mathbf{G}\mathbf{H}\boldsymbol{z} + \mathbf{G}\boldsymbol{\Sigma}\mathbf{G}^T(\mathbf{G}\boldsymbol{\Sigma}\mathbf{G}^T)^{-1}\boldsymbol{r} - \mathbf{G}\boldsymbol{\Sigma}\mathbf{G}^T(\mathbf{M}\boldsymbol{z}_2) = \boldsymbol{r},$$

which means $\mathbf{G}\boldsymbol{\Sigma}\mathbf{G}^T(\mathbf{M}\boldsymbol{z}_2) = \mathbf{G}\mathbf{H}\boldsymbol{z}$. Thus we have

$$\boldsymbol{x} = \mathbf{H}\boldsymbol{z} + \boldsymbol{\Sigma}\mathbf{G}^T(\mathbf{G}\boldsymbol{\Sigma}\mathbf{G}^T)^{-1}(\boldsymbol{r} - \mathbf{G}\mathbf{H}\boldsymbol{z}).$$

In addition, since if $\boldsymbol{z} \sim \mathcal{N}[\mathbf{H}^{-1}\boldsymbol{\mu}, \mathbf{H}^{-1}\boldsymbol{\Sigma}(\mathbf{H}^{-1})^T]$, then $\boldsymbol{y} = \mathbf{H}\boldsymbol{z} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$. Therefore, without the need to compute any intermediate variables, one may use Algorithm 2 to generate $\boldsymbol{x}$ from the hyperplane truncated MVN distribution.

□

*Proof of Theorem 2.* Using the matrix inversion lemma on (8), we have

$$\begin{aligned} p(\boldsymbol{x}_1) &\propto \exp\left[-\frac{1}{2}(\boldsymbol{x}_1 - \boldsymbol{\mu}_1)^T \left(\boldsymbol{\Sigma}_{11} - \boldsymbol{\Sigma}_{12}\boldsymbol{\Sigma}_{22}^{-1}\boldsymbol{\Sigma}_{21}\right)^{-1}(\boldsymbol{x}_1 - \boldsymbol{\mu}_1)\right], \\ &\propto \exp\left\{-\frac{1}{2}(\boldsymbol{x}_1 - \boldsymbol{\mu}_1)^T \left[\boldsymbol{\Sigma}_{11}^{-1} + \boldsymbol{\Sigma}_{11}^{-1}\boldsymbol{\Sigma}_{12}\left(\boldsymbol{\Sigma}_{22} - \boldsymbol{\Sigma}_{21}\boldsymbol{\Sigma}_{11}^{-1}\boldsymbol{\Sigma}_{12}\right)^{-1}\boldsymbol{\Sigma}_{21}\boldsymbol{\Sigma}_{11}^{-1}\right](\boldsymbol{x}_1 - \boldsymbol{\mu}_1)\right\}. \end{aligned} \tag{21}$$

Using (11), we have $\mathbf{G}\boldsymbol{\mu} = \mathbf{G}_1\boldsymbol{\mu}_1 + \mathbf{G}_2\boldsymbol{\mu}_2 = \boldsymbol{r}$. Since $\mathbf{G}\boldsymbol{x} = \boldsymbol{r}$, we further have $\mathbf{G}(\boldsymbol{x} - \boldsymbol{\mu}) = 0$ and hence $\mathbf{G}_1(\boldsymbol{x}_1 - \boldsymbol{\mu}_1) = -\mathbf{G}_2(\boldsymbol{x}_2 - \boldsymbol{\mu}_2)$. Therefore, given the construction of $\boldsymbol{\mu}_2$ as in (11), we can replace the equality constraint $\mathbf{G}\boldsymbol{x} = \boldsymbol{r}$ on $\boldsymbol{x}$ by requiring $(\boldsymbol{x}_2 - \boldsymbol{\mu}_2) = -\mathbf{G}_2^{-1}\mathbf{G}_1(\boldsymbol{x}_1 - \boldsymbol{\mu}_1)$. Using this equivelent constraint together with (3), we have

$$\begin{aligned} & p(\boldsymbol{x} \,|\, \boldsymbol{\mu}, \tilde{\boldsymbol{\Sigma}}, \mathbf{G}, \boldsymbol{r}) \\ \propto & \exp\left[-\frac{1}{2}(\boldsymbol{x}_1 - \boldsymbol{\mu}_1)^T\boldsymbol{\Sigma}_{11}^{-1}(\boldsymbol{x}_1 - \boldsymbol{\mu}_1) - \frac{1}{2}(\boldsymbol{x}_2 - \boldsymbol{\mu}_2)^T\tilde{\boldsymbol{\Sigma}}_{22}^{-1}(\boldsymbol{x}_2 - \tilde{\boldsymbol{\mu}}_2)\right]\delta(\mathbf{G}\boldsymbol{x} = \boldsymbol{r}) \\ \propto & \exp\left\{-\frac{1}{2}(\boldsymbol{x}_1 - \boldsymbol{\mu}_1)^T\left[\boldsymbol{\Sigma}_{11}^{-1} + \mathbf{G}_1^T(\mathbf{G}_2^{-1})^T\tilde{\boldsymbol{\Sigma}}_{22}^{-1}\mathbf{G}_2^{-1}\mathbf{G}_1\right](\boldsymbol{x}_1 - \boldsymbol{\mu}_1)\right\} \\ & \times \delta\left[\boldsymbol{x}_2 = \boldsymbol{\mu}_2 - \mathbf{G}_2^{-1}\mathbf{G}_1(\boldsymbol{x}_1 - \boldsymbol{\mu}_1)\right] \\ = & \mathcal{N}\left\{\boldsymbol{x}_1; \boldsymbol{\mu}_1, \left[\boldsymbol{\Sigma}_{11}^{-1} + \mathbf{G}_1^T(\mathbf{G}_2^{-1})^T\tilde{\boldsymbol{\Sigma}}_{22}^{-1}\mathbf{G}_2^{-1}\mathbf{G}_1\right]^{-1}\right\} \\ & \times \delta\left[\boldsymbol{x}_2 = \boldsymbol{\mu}_2 - \mathbf{G}_2^{-1}\mathbf{G}_1(\boldsymbol{x}_1 - \boldsymbol{\mu}_1)\right] \end{aligned} \tag{22}$$

It is clear that the marginal distribution of $\boldsymbol{x}_1$ in (22) matches the conditional distribution of $\boldsymbol{x}_1$ in (21) if we further construct $\tilde{\boldsymbol{\Sigma}}_{22}$ using (12). □

*Proof of Corollary 4.* Applying Theorem 1 to Corollary 3, we can generate $\boldsymbol{x}$ with

- Sample $\boldsymbol{y} \sim \mathcal{N}\left(\boldsymbol{\mu}, \tilde{\boldsymbol{\Sigma}}\right)$;
- Return $\boldsymbol{x} = \boldsymbol{y} + \tilde{\boldsymbol{\Sigma}}\mathbf{G}^T[\mathbf{G}\tilde{\boldsymbol{\Sigma}}\mathbf{G}^T]^{-1}(\boldsymbol{r} - \mathbf{G}\boldsymbol{y})$.

Since $\boldsymbol{\mu} = \begin{bmatrix}\boldsymbol{\mu}_1 \\ \mathbf{0}\end{bmatrix}$, $\tilde{\boldsymbol{\Sigma}} = \begin{bmatrix}\boldsymbol{\Sigma}_{11} & \mathbf{0} \\ \mathbf{0} & \boldsymbol{\Sigma}_{22} - \boldsymbol{\Sigma}_{21}\boldsymbol{\Sigma}_{11}^{-1}\boldsymbol{\Sigma}_{12}\end{bmatrix}$, $\mathbf{G} = (\mathbf{G}_1, \mathbf{G}_2) = (\boldsymbol{\Sigma}_{21}\boldsymbol{\Sigma}_{11}^{-1}, \mathbf{I}_{k_2})$, and $\boldsymbol{r} = \boldsymbol{\Sigma}_{21}\boldsymbol{\Sigma}_{11}^{-1}\boldsymbol{\mu}_1$, we have $\tilde{\boldsymbol{\Sigma}}\mathbf{G}^T = \begin{bmatrix}\boldsymbol{\Sigma}_{12} \\ \boldsymbol{\Sigma}_{22} - \boldsymbol{\Sigma}_{21}\boldsymbol{\Sigma}_{11}^{-1}\boldsymbol{\Sigma}_{12}\end{bmatrix}$ and $[\mathbf{G}\tilde{\boldsymbol{\Sigma}}\mathbf{G}^T]^{-1} = \boldsymbol{\Sigma}_{22}^{-1}$. Since $\tilde{\boldsymbol{\Sigma}}$ is block diagonal, we can independently sample $\boldsymbol{y}_1$ and $\boldsymbol{y}_2$ as $\boldsymbol{y}_1 \sim \mathcal{N}(\boldsymbol{\mu}_1, \boldsymbol{\Sigma}_{11})$ and $\boldsymbol{y}_2 \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma}_{22} - \boldsymbol{\Sigma}_{21}\boldsymbol{\Sigma}_{11}^{-1}\boldsymbol{\Sigma}_{12})$, respectively, with which we can further sample $\boldsymbol{x}$ as

$$\begin{bmatrix}\boldsymbol{x}_1 \\ \boldsymbol{x}_2\end{bmatrix} = \begin{bmatrix}\boldsymbol{y}_1 \\ \boldsymbol{y}_2\end{bmatrix} + \begin{bmatrix}\boldsymbol{\Sigma}_{12} \\ \boldsymbol{\Sigma}_{22} - \boldsymbol{\Sigma}_{21}\boldsymbol{\Sigma}_{11}^{-1}\boldsymbol{\Sigma}_{12}\end{bmatrix}\boldsymbol{\Sigma}_{22}^{-1}(\boldsymbol{\Sigma}_{21}\boldsymbol{\Sigma}_{11}^{-1}\boldsymbol{\mu}_1 - \boldsymbol{\Sigma}_{21}\boldsymbol{\Sigma}_{11}^{-1}\boldsymbol{y}_1 - \boldsymbol{y}_2).$$

Thus we can let $\boldsymbol{x}_1 = \boldsymbol{\mu}_1 + \boldsymbol{y}_1' - \boldsymbol{\Sigma}_{12}\boldsymbol{\Sigma}_{22}^{-1}\left(\boldsymbol{\Sigma}_{21}\boldsymbol{\Sigma}_{11}^{-1}\boldsymbol{y}_1' + \boldsymbol{y}_2\right)$, where $\boldsymbol{y}_1' = \boldsymbol{y}_1 - \boldsymbol{\mu}_1 \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma}_{11})$. □

*Proof of Corollary 5.* Using the matrix inversion lemma, we have

$$\boldsymbol{\Sigma}_\beta = \mathbf{A}^{-1} - \mathbf{A}^{-1}\boldsymbol{\Phi}^T(\boldsymbol{\Omega}^{-1} + \boldsymbol{\Phi}\mathbf{A}^{-1}\boldsymbol{\Phi}^T)^{-1}\boldsymbol{\Phi}\mathbf{A}^{-1}. \tag{23}$$

The proof is completed by using Corollary 4 with $\boldsymbol{\mu}_1 = \boldsymbol{\mu}_\beta$, $\boldsymbol{\Sigma}_{11} = \mathbf{A}^{-1}$, $\boldsymbol{\Sigma}_{12} = \mathbf{A}^{-1}\boldsymbol{\Phi}^T$, and $\boldsymbol{\Sigma}_{22} = \boldsymbol{\Omega}^{-1} + \boldsymbol{\Phi}\mathbf{A}^{-1}\boldsymbol{\Phi}^T$. □

## Computational Complexity

We present the computational complexities of all proposed algorithms in the following tables, where we highlight with bold the lowest complexity in each row.

Table 1: Computational complexity of Algorithm 1.

| Calculation | Computational complexity | |
|---|---|---|
| | Non-diagonal $\boldsymbol{\Sigma}$ | Diagonal $\boldsymbol{\Sigma}$ |
| $\mathbf{H}$ | $\mathcal{O}(k_2k^2)$ | $\mathcal{O}(k_2k^2)$ |
| $\boldsymbol{z}_2$ | $\mathcal{O}(k_2^2k)$ | $\mathcal{O}(k_2^2k)$ |
| $\boldsymbol{\Sigma}^{-1}$ | $\mathcal{O}(k^3)$ | $\boldsymbol{\mathcal{O}(k)}$ |
| $\boldsymbol{\Lambda}_{11}$ | $\mathcal{O}(k_1k^2)$ | $\boldsymbol{\mathcal{O}(k_1^2k)}$ |
| $\boldsymbol{\Lambda}_{12}$ | $\mathcal{O}(k_1k_2k)$ | $\mathcal{O}(k_1k_2k)$ |
| $\boldsymbol{\mu}_{\boldsymbol{z}_1}$ | $\mathcal{O}(\max(k^2, k_1^3, k_1^2k_2))$ | $\mathcal{O}(\max(k^2, k_1^3, k_1^2k_2))$ |
| $\boldsymbol{z}_1$ | $\mathcal{O}(k_1^3)$ | $\mathcal{O}(k_1^3)$ |
| $\boldsymbol{x}$ | $\mathcal{O}(\max(k_1k, k_2k))$ | $\mathcal{O}(\max(k_1k, k_2k))$ |
| Summary | $\mathcal{O}(k^3)$ | $\boldsymbol{\mathcal{O}(\max(k_2k^2, k_1^2k))}$ |

Table 2: Computational complexity of Algorithm 2.

| Calculation | Computational complexity | |
|---|---|---|
| | Non-diagonal $\boldsymbol{\Sigma}$ | Diagonal $\boldsymbol{\Sigma}$ |
| $\boldsymbol{y}$ | $\mathcal{O}(k^3)$ | $\boldsymbol{\mathcal{O}(k)}$ |
| $\mathbf{G}\boldsymbol{\Sigma}\mathbf{G}^T$ | $\mathcal{O}(k_2k^2)$ | $\boldsymbol{\mathcal{O}(k_2^2k)}$ |
| $\boldsymbol{\alpha}$ | $\mathcal{O}(\max(k_2k, k_2^3))$ | $\mathcal{O}(\max(k_2k, k_2^3))$ |
| $\boldsymbol{x}$ | $\mathcal{O}(k_2k)$ | $\mathcal{O}(k_2k)$ |
| Summary | $\mathcal{O}(k^3)$ | $\boldsymbol{\mathcal{O}(k_2^2k)}$ |

Table 3: Computational complexity of naive simulation in Algorithm 3.

| Calculation | Computational complexity | | | |
|---|---|---|---|---|
| | Non-diagonal $\boldsymbol{\Sigma}_{11}$<br>Non-diagonal $\boldsymbol{\Sigma}_{22}$ | Diagonal $\boldsymbol{\Sigma}_{11}$<br>Non-diagonal $\boldsymbol{\Sigma}_{22}$ | Non-diagonal $\boldsymbol{\Sigma}_{11}$<br>Diagonal $\boldsymbol{\Sigma}_{22}$ | Diagonal $\boldsymbol{\Sigma}_{11}$<br>Diagonal $\boldsymbol{\Sigma}_{22}$ |
| $\boldsymbol{\Sigma}_{22}^{-1}$ | $\mathcal{O}(k_2^3)$ | $\mathcal{O}(k_2^3)$ | $\boldsymbol{\mathcal{O}(k_2)}$ | $\boldsymbol{\mathcal{O}(k_2)}$ |
| $\boldsymbol{\Sigma}_{12}\boldsymbol{\Sigma}_{22}^{-1}\boldsymbol{\Sigma}_{21}$ | $\mathcal{O}(\max(k_1^2k_2, k_1k_2^2))$ | $\mathcal{O}(\max(k_1^2k_2, k_1k_2^2))$ | $\boldsymbol{\mathcal{O}(k_1^2k_2)}$ | $\boldsymbol{\mathcal{O}(k_1^2k_2)}$ |
| $\boldsymbol{x}_1$ | $\mathcal{O}(k_1^3)$ | $\mathcal{O}(k_1^3)$ | $\mathcal{O}(k_1^3)$ | $\mathcal{O}(k_1^3)$ |
| Summary | $\mathcal{O}(\max(k_1^3, k_2^3))$ | $\mathcal{O}(\max(k_1^3, k_2^3))$ | $\boldsymbol{\mathcal{O}(\max(k_1^3, k_1^2k_2))}$ | $\boldsymbol{\mathcal{O}(\max(k_1^3, k_1^2k_2))}$ |

Table 4: Computational complexity of Algorithm 3.

| Calculation | Computational complexity | | | |
|---|---|---|---|---|
| | Non-diagonal $\boldsymbol{\Sigma}_{11}$<br>Non-diagonal $\boldsymbol{\Sigma}_{22}$ | Diagonal $\boldsymbol{\Sigma}_{11}$<br>Non-diagonal $\boldsymbol{\Sigma}_{22}$ | Non-diagonal $\boldsymbol{\Sigma}_{11}$<br>Diagonal $\boldsymbol{\Sigma}_{22}$ | Diagonal $\boldsymbol{\Sigma}_{11}$<br>Diagonal $\boldsymbol{\Sigma}_{22}$ |
| $\boldsymbol{y}_1$ | $\mathcal{O}(k_1^3)$ | $\boldsymbol{\mathcal{O}(k_1)}$ | $\mathcal{O}(k_1^3)$ | $\boldsymbol{\mathcal{O}(k_1)}$ |
| $\boldsymbol{\Sigma}_{11}^{-1}$ | $\mathcal{O}(k_1^3)$ | $\boldsymbol{\mathcal{O}(k_1)}$ | $\mathcal{O}(k_1^3)$ | $\boldsymbol{\mathcal{O}(k_1)}$ |
| $\boldsymbol{\Sigma}_{21}\boldsymbol{\Sigma}_{11}^{-1}\boldsymbol{\Sigma}_{12}$ | $\mathcal{O}(\max(k_1k_2^2, k_1^2k_2))$ | $\boldsymbol{\mathcal{O}(k_1k_2^2)}$ | $\mathcal{O}(\max(k_1k_2^2, k_1^2k_2))$ | $\boldsymbol{\mathcal{O}(k_1k_2^2)}$ |
| $\boldsymbol{y}_2$ | $\mathcal{O}(k_2^3)$ | $\mathcal{O}(k_2^3)$ | $\mathcal{O}(k_2^3)$ | $\mathcal{O}(k_2^3)$ |
| $\boldsymbol{\alpha}$ | $\mathcal{O}(\max(k_1k_2, k_2^3))$ | $\mathcal{O}(\max(k_1k_2, k_2^3))$ | $\boldsymbol{\mathcal{O}(k_1k_2)}$ | $\boldsymbol{\mathcal{O}(k_1k_2)}$ |
| $\boldsymbol{x}_1$ | $\mathcal{O}(k_1k_2)$ | $\mathcal{O}(k_1k_2)$ | $\mathcal{O}(k_1k_2)$ | $\mathcal{O}(k_1k_2)$ |
| Summary | $\mathcal{O}(\max(k_1^3, k_2^3))$ | $\boldsymbol{\mathcal{O}(\max(k_1k_2^2, k_2^3))}$ | $\mathcal{O}(\max(k_1^3, k_2^3))$ | $\boldsymbol{\mathcal{O}(\max(k_1k_2^2, k_2^3))}$ |

Table 5: Computational complexity of naive simulation in Algorithm 4.

| Calculation | Computational complexity | | | |
|---|---|---|---|---|
| | Non-diagonal $\mathbf{A}$<br>Non-diagonal $\boldsymbol{\Omega}$ | Diagonal $\mathbf{A}$<br>Non-diagonal $\boldsymbol{\Omega}$ | Non-diagonal $\mathbf{A}$<br>Diagonal $\boldsymbol{\Omega}$ | Diagonal $\mathbf{A}$<br>Diagonal $\boldsymbol{\Omega}$ |
| $\boldsymbol{\Phi}^T\boldsymbol{\Omega}\boldsymbol{\Phi}$ | $\mathcal{O}(\max(n^2p, np^2))$ | $\mathcal{O}(\max(n^2p, np^2))$ | $\boldsymbol{\mathcal{O}(np^2)}$ | $\boldsymbol{\mathcal{O}(np^2)}$ |
| $(\mathbf{A} + \boldsymbol{\Phi}^T\boldsymbol{\Omega}\boldsymbol{\Phi})^{-1}$ | $\mathcal{O}(p^3)$ | $\mathcal{O}(p^3)$ | $\mathcal{O}(p^3)$ | $\mathcal{O}(p^3)$ |
| $\boldsymbol{\beta}$ | $\mathcal{O}(p^3)$ | $\mathcal{O}(p^3)$ | $\mathcal{O}(p^3)$ | $\mathcal{O}(p^3)$ |
| Summary | $\mathcal{O}(\max(n^2p, p^3))$ | $\mathcal{O}(\max(n^2p, p^3))$ | $\boldsymbol{\mathcal{O}(\max(np^2, p^3))}$ | $\boldsymbol{\mathcal{O}(\max(np^2, p^3))}$ |

Table 6: Computational complexity of Algorithm 4.

| Calculation | Computational complexity | | | |
|---|---|---|---|---|
| | Non-diagonal $\mathbf{A}$<br>Non-diagonal $\mathbf{\Omega}$ | Diagonal $\mathbf{A}$<br>Non-diagonal $\mathbf{\Omega}$ | Non-diagonal $\mathbf{A}$<br>Diagonal $\mathbf{\Omega}$ | Diagonal $\mathbf{A}$<br>Diagonal $\mathbf{\Omega}$ |
| $\mathbf{A}^{-1}$ | $\mathcal{O}(p^3)$ | $\boldsymbol{\mathcal{O}(p)}$ | $\mathcal{O}(p^3)$ | $\boldsymbol{\mathcal{O}(p)}$ |
| $\boldsymbol{y}_1$ | $\mathcal{O}(p^3)$ | $\boldsymbol{\mathcal{O}(p)}$ | $\mathcal{O}(p^3)$ | $\boldsymbol{\mathcal{O}(p)}$ |
| $\mathbf{\Omega}^{-1}$ | $\mathcal{O}(n^3)$ | $\mathcal{O}(n^3)$ | $\boldsymbol{\mathcal{O}(n)}$ | $\boldsymbol{\mathcal{O}(n)}$ |
| $\boldsymbol{y}_2$ | $\mathcal{O}(n^3)$ | $\mathcal{O}(n^3)$ | $\boldsymbol{\mathcal{O}(n)}$ | $\boldsymbol{\mathcal{O}(n)}$ |
| $\mathbf{\Omega}^{-1}+\mathbf{\Phi}\mathbf{A}^{-1}\mathbf{\Phi}^T$ | $\mathcal{O}(\max(np^2,n^2p))$ | $\boldsymbol{\mathcal{O}(n^2p)}$ | $\mathcal{O}(\max(np^2,n^2p))$ | $\boldsymbol{\mathcal{O}(n^2p)}$ |
| $\boldsymbol{\alpha}$ | $\mathcal{O}(\max(np,n^3))$ | $\mathcal{O}(\max(np,n^3))$ | $\mathcal{O}(\max(np,n^3))$ | $\mathcal{O}(\max(np,n^3))$ |
| $\boldsymbol{\beta}$ | $\mathcal{O}(np)$ | $\mathcal{O}(np)$ | $\mathcal{O}(np)$ | $\mathcal{O}(np)$ |
| Summary | $\mathcal{O}(\max(n^3,p^3))$ | $\boldsymbol{\mathcal{O}(\max(n^2p,n^3))}$ | $\mathcal{O}(\max(n^3,p^3))$ | $\boldsymbol{\mathcal{O}(\max(n^2p,n^3))}$ |

## Brief derivation of SG-MCMC for a simplex-constrained vector

Based on a comprehensive framework for constructing SG-MCMC algorithms in Ma et al. (2015), we have the mini-batch update rule for a global variable $\boldsymbol{z}$ as

$$\begin{aligned}\boldsymbol{z}_{t+1}=\boldsymbol{z}_t+\varepsilon_t\Big\{-\big[\mathbf{D}(\boldsymbol{z}_t)+\mathbf{Q}(\boldsymbol{z}_t)\big]\nabla\tilde{H}(\boldsymbol{z}_t)+\Gamma(\boldsymbol{z}_t)\Big\}\\ +\mathcal{N}\left(\mathbf{0},\varepsilon_t\left[2\mathbf{D}\left(\boldsymbol{z}_t\right)-\varepsilon_t\hat{\mathbf{B}}_t\right]\right),\end{aligned}\tag{24}$$

where $\varepsilon_t$ are annealed step sizes, $\mathbf{D}\left(\boldsymbol{z}\right)$ is a positive semidefinite diffusion matrix, $\mathbf{Q}\left(\boldsymbol{z}\right)$ is a skew-symmetric curl matrix, $\hat{\mathbf{B}}_t$ is an estimate of the stochastic gradient noise variance satisfying a positive definite constraint as $2\mathbf{D}\left(\boldsymbol{z}_t\right)-\varepsilon_t\hat{\mathbf{B}}_t\succ\mathbf{0}$, and $\Gamma_i(\boldsymbol{z})$, the $i^{th}$ element of the compensation vector $\Gamma(\boldsymbol{z})$, is defined as $\Gamma_i\left(\boldsymbol{z}\right)=\sum_j\frac{\partial}{\partial z_j}\left[\mathbf{D}_{ij}\left(\boldsymbol{z}\right)+\mathbf{Q}_{ij}\left(\boldsymbol{z}\right)\right]$. The mini-batch estimation of energy function is defined as $\tilde{H}\left(\boldsymbol{z}\right)=-\ln p\left(\boldsymbol{z}\right)-\rho\sum_{\boldsymbol{x}\in\tilde{X}}\ln p\left(\boldsymbol{x}\,|\boldsymbol{z}\right)$, with $\tilde{X}$ the mini-batch and $\rho$ the ratio of the dataset size to the mini-batch size.

For simplicity, we adopt the same specifications that lead to the stochastic gradient Riemannian Langevin dynamics (SGRLD) inference algorithm for simplex-constrained model parameters (Patterson and Teh 2013; Ma et al. 2015), namely $\mathbf{D}\left(\boldsymbol{z}\right)=\mathbf{G}(\boldsymbol{z})^{-1}$, $\mathbf{Q}\left(\boldsymbol{z}\right)=\mathbf{0}$, and $\hat{\mathbf{B}}_t=\mathbf{0}$, where $\mathbf{G}\left(\boldsymbol{z}\right)$ denotes the Fisher information matrix (FIM) (Girolami and Calderhead 2011).

With the multinomial likelihood $\boldsymbol{n}_j\sim\mathrm{Mult}\left(n_{\cdot j},\boldsymbol{\phi}\right)$, and the reduced-mean parameterization $\boldsymbol{\varphi}\in\mathbb{R}_+^{V-1}$, where $j\in\{1,\cdots,N\}$ with $N$ the dataset size, it is straight to derive the FIM as

$$\begin{aligned}\mathbf{G}\left(\boldsymbol{\varphi}\right)&=-\mathbb{E}\left[\frac{\partial^2}{\partial\boldsymbol{\varphi}^2}\ln\left(\prod\nolimits_j\mathrm{Mult}\left[\boldsymbol{n}_j;n_{\cdot j},\boldsymbol{\phi}\right]\right)\right]\\&=M\left[\mathrm{diag}\left(1/\boldsymbol{\varphi}\right)+\mathbf{1}\mathbf{1}^T/(1-\varphi_\cdot)\right],\end{aligned}\tag{25}$$

where $M:=\mathbb{E}\left[\sum_{j=1}^N n_{\cdot j}\right]$ is approximated along the updating as $M=(1-\varepsilon_t)M+\varepsilon_t\rho E\left[n_{\cdot\cdot}\right]$. Further with the Dirichlet prior $\boldsymbol{\phi}\sim\mathrm{Dir}\left(\eta\mathbf{1}_V\right)$, we have the conditional

posterior of $\boldsymbol{\phi}$ as $(\boldsymbol{\phi}|-) \sim \text{Dir}(\sum_j n_{1j}+\eta, \ldots, \sum_j n_{Vj}+\eta)$. Taking the gradient with respect to the reduced-mean parameterization $\boldsymbol{\varphi} \in \mathbb{R}_+^{V-1}$ on the mini-batch estimation of the negative energy function, we have

$$\nabla_{\boldsymbol{\varphi}}\Big[-\tilde{H}(\boldsymbol{\varphi})\Big] = \frac{\rho\bar{\boldsymbol{n}}_{:.}+\eta-1}{\boldsymbol{\varphi}} - \frac{\rho n_{V.}+\eta-1}{1-\varphi_.}. \tag{26}$$

Substituting both (25) and (26) into (24), we have (16) as

$$\boldsymbol{\varphi}_{t+1} = \left[\boldsymbol{\varphi}_t + \frac{\varepsilon_t}{M}[(\rho\bar{\boldsymbol{n}}_{:.}+\eta) - (\rho n_{..}+\eta V)\,\boldsymbol{\varphi}_t] + \mathcal{N}\left(\mathbf{0}, \frac{2\varepsilon_t}{M}\left[\text{diag}\,(\boldsymbol{\varphi}_t) - \boldsymbol{\varphi}_t\boldsymbol{\varphi}_t^T\right]\right)\right]_{\triangle},$$

where $[\cdot]_{\triangle}$, denoting the constraint that $\boldsymbol{\varphi} \in \mathbb{R}_+^{V-1}$ and $\varphi_. \leq 1$, ensures $\boldsymbol{\varphi}$ to be valid.

Next we prove that equation (16) can be equivalently represented as (17), namely

$$\boldsymbol{\phi}_{t+1} = \left[\boldsymbol{\phi}_t + \frac{\varepsilon_t}{M}[(\rho\boldsymbol{n}_{:.}+\eta) - (\rho n_{..}+\eta V)\,\boldsymbol{\phi}_t] + \mathcal{N}\left(\mathbf{0}, \frac{2\varepsilon_t}{M}\text{diag}\,(\boldsymbol{\phi}_t)\right)\right]_{\angle},$$

where $[\cdot]_{\angle}$ represents the constraint that $\boldsymbol{\phi} \in \mathbb{R}_+^V$ and $\mathbf{1}^T\boldsymbol{\phi} = 1$. By substituting $\boldsymbol{\phi} = (\boldsymbol{\varphi}^T, 1-\mathbf{1}^T\boldsymbol{\varphi})^T$ into (17), one can easily verify that the MVN simulation in (17) is identical to that in (16). By further pointing out the fact that $[\cdot]_{\triangle}$ is the same as $[\cdot]_{\angle}$ under the reduced-mean parameterization, we conclude the proof.