Stochastic Newton Sampler: The R Package sns

Alireza S. Mahani
Scientific Computing
Sentrana Inc.

Asad Hasan
Scientific Computing
Sentrana Inc.

Marshall Jiang
Department of Mathematics
Cornell University

Mansour T.A. Sharabiani
National Heart and Lung Institute
Imperial College London

Abstract

The R package **sns** implements Stochastic Newton Sampler (SNS), a Metropolis-Hastings Monte Carlo Markov Chain algorithm where the proposal density function is a multivariate Gaussian based on a local, second-order Taylor-series expansion of log-density. The mean of the proposal function is the full Newton step in Newton-Raphson optimization algorithm. Taking advantage of the local, multivariate geometry captured in log-density Hessian allows SNS to be more efficient than univariate samplers, approaching independent sampling as the density function increasingly resembles a multivariate Gaussian. SNS requires the log-density Hessian to be negative-definite everywhere in order to construct a valid proposal function. This property holds, or can be easily checked, for many GLM-like models. When initial point is far from density peak, running SNS in non-stochastic mode by taking the Newton step - augmented with line search - allows the MCMC chain to converge to high-density areas faster. For high-dimensional problems, partitioning of state space into lower-dimensional subsets, and applying SNS to the subsets within a Gibbs sampling framework can significantly improve the mixing of SNS chains. In addition to the above strategies for improving convergence and mixing, **sns** offers utilities for diagnostics and visualization, sample-based calculation of Bayesian predictive posterior distributions, numerical differentiation, and log-density validation.

Keywords: monte carlo markov chain, metropolis-hastings, newton-raphson optimization, negative-definite hessian, log-concavity.

1. Introduction

In most real-world applications of Monte Carlo Markov Chain (MCMC) sampling, the probability density function (PDF) being sampled is multidimensional. Univariate samplers such as the slice sampler (Neal 2003) and adaptive rejection sampler (Gilks and Wild 1992) can be embedded in the Gibbs sampling framework (Geman and Geman 1984) to sample from multivariate PDFs (Mahani and Sharabiani 2015a). Univariate samplers generally have few tuning parameters, making them ideal candidates for black-box MCMC software such as JAGS (Plummer 2013) and OpenBUGS (Thomas, O'Hara, Ligges, and Sturtz 2006). However, they become less effective as PDF dimensionality rises and dimensions become more

correlated (Girolami and Calderhead 2011). Therefore, development - and software implementation - of efficient, black-box multivariate MCMC algorithms is of great importance to widespread application of probabilistic models in statistics and machine learning. Recent theoretical (Girolami and Calderhead 2011; Hoffman and Gelman 2014) and software development (Stan Development Team 2014) efforts to make multivariate samplers such as Hamiltonian Monte Carlo (HMC) (Duane *et al.* 1987) more self-tuning and efficient can be viewed in this light.

The R package **sns** implements Stochastic Newton Sampler (SNS), a Metropolis-Hastings MCMC algorithm (Hastings 1970), where the proposal distribution is a locally-fitted multivariate Gaussian resulting from second-order Taylor series expansion of the log-density. In its current implementation, SNS requires the log-density to be twice-diffenrentiable and globally concave, or equivalently that its Hessian matrix be negative-definite everywhere. For many Generalized Linear Models (GLMs) these conditions are satisfied (Gilks and Wild 1992), and the invariance theorem of Section 2.3 allows Hessian negative-definiteness to be studied and proven in the much lower-dimensional space of linear predictors, rather than the high-dimensional space of regression coefficients.

SNS has appeared in the literature under several variations and labels. Gamerman (1997) extend Iterative Reweighted Least Squares (IRLS) - the primary estimation technique for GLM models - to MCMC sampling by adding a Metropolis-Hastings step to it. Given that IRLS is a close cousin of Newton-Raphson optimization, their method can be considered a specialization of SNS for GLM models. Qi and Minka (2002) present what they call 'Hessian-based Metropolis-Hastings' (HMH), which is nearly identical to SNS, but they do not address the high-dimensional mixing problem, nor do they provide an open-source software implementation. More recently, the simplified Manifold Metropolis Adjusted Langevin Algorithm (MMALA) proposed in Girolami and Calderhead (2011) is very similar to SNS with the addition of a tunable step size, or learning rate. The software accompanying their paper, however, is written in MATLAB (MATLAB 2014). The R package **sns**, to our knowledge, is the first open-source implemnentation of the SNS algorithm, including extensions for improving convergence (`rnd` and `nnr` arguments) and mixing (`part` argument), diagnostic and visualization methods (`summary.sns` and `plot.sns`), sample-based prediction (`predict.sns`), and numerical differentiation using **numDeriv** package (Gilbert and Varadhan 2012).

The paper is organized as follows. In Section 2, we review the theoretical background for SNS, including assessment of log-concavity and calculation of gradient and Hessian. In Section 3, we discuss the implementation of SNS algorithm in the **sns** package, including facilities for improving convergence and mixing, full Bayesian prediction, and numerical calculation of PDF derivatives. Section 4 offers several examples to illustrate the usage of **sns** features, as well as how SNS can be combined with other samplers. In Section 5 we study the performance of SNS in comparison with a few other samplers. Finally, Section 6 provides a summary of our work and a discussion of potential future research and development directions.

2. Theory

In this section, we provide a brief overview of Metropolis-Hastings algorithm and offer a formal definition of the SNS proposal PDF. This is followed by a discussion of log-concavity requirement as well as computation of gradient and Hessian, focusing on an important class

of models, referred to as 'exchangeable regression' models (Section 2.3).

2.1. Metropolis-Hastings algorithm

In Metropolis-Hastings (MH) MCMC sampling of the PDF, $p(\mathbf{z})$, we generate a sample $\mathbf{z}^*$ from the proposal density function $q(\mathbf{z}|\mathbf{z}^\tau)$, where $\mathbf{z}^\tau$ is the current state. We then accept the proposed state $\mathbf{z}^*$ with probability $A(\mathbf{z}^*, \mathbf{z}^\tau)$, where:

$$A(\mathbf{z}^*, \mathbf{z}^\tau) = \min\left(1\,,\, \frac{p(\mathbf{z}^*)q(\mathbf{z}^\tau|\mathbf{z}^*)}{p(\mathbf{z}^\tau)q(\mathbf{z}^*|\mathbf{z}^\tau)}\right) \tag{1}$$

The MH transitions satisfy detailed balance:

$$\begin{aligned} p(\mathbf{z}^\tau)q(\mathbf{z}^*|\mathbf{z}^\tau)A(\mathbf{z}^*, \mathbf{z}^\tau) &= \min\left(p(\mathbf{z}^\tau)q(\mathbf{z}^*|\mathbf{z}^\tau)\,,\, p(\mathbf{z}^*)q(\mathbf{z}^\tau|\mathbf{z}^*)\right) \\ &= \min\left(p(\mathbf{z}^*)q(\mathbf{z}^\tau|\mathbf{z}^*)\,,\, p(\mathbf{z}^\tau)q(\mathbf{z}^*|\mathbf{z}^\tau)\right) \\ &= p(\mathbf{z}^*)q(\mathbf{z}^\tau|\mathbf{z}^*)A(\mathbf{z}^\tau, \mathbf{z}^*) \end{aligned} \tag{2}$$

The detailed balance property ensures that $p(\mathbf{z})$ is invariant under MH transitions. For a discussion of ergodicity of MH algorithm, see Roberts and Rosenthal (1999).

2.2. SNS proposal density

SNS proposal density is a multivariate Gaussian fitted locally to the density being sampled, using the second-order Taylor-series expansion of the log-density:

$$f(\mathbf{x}) \approx f(\mathbf{x}_0) + \mathbf{g}(\mathbf{x}_0)^\top (\mathbf{x} - \mathbf{x}_0) + \frac{1}{2}(\mathbf{x} - \mathbf{x}_0)^\top \mathbf{H}(\mathbf{x}_0)\,(\mathbf{x} - \mathbf{x}_0) \tag{3}$$

where $f : \mathbb{R}^K \to \mathbb{R}$ is the log-density, and $\mathbf{g}$ and $\mathbf{H}$ are the gradient vector and Hessian matrix for f, respectively, with dimensions K and $K \times K$. Assuming that f is globally concave, the above approximation is equivalent to fitting the following multivariate Gaussian (which we refer to as $F(\mathbf{x})$) to the PDF:

$$F(\mathbf{x}) = \frac{1}{(2\pi)^{K/2}|\mathbf{\Sigma}|^{1/2}} \exp\left\{-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu})^T \mathbf{\Sigma}^{-1}(\mathbf{x} - \boldsymbol{\mu})\right\} \tag{4}$$

By comparing Equations 3 and 4, we see that the precision matrix is the same as negative Hessian: $\mathbf{\Sigma}^{-1} = -\mathbf{H}(\mathbf{x}_0)$. The mean of the fitted Gaussian maximizes its log, and therefore:

$$\boldsymbol{\mu} = \mathbf{x}_0 - \mathbf{H}^{-1}(\mathbf{x}_0)\,\mathbf{g}(\mathbf{x}_0) \tag{5}$$

We can now formally define the (multivariate Gaussian) proposal density $q(.\,|\,\mathbf{x})$ as:

$$q(.\,|\,\mathbf{x}) = \mathcal{N}(\mathbf{x} - \mathbf{H}^{-1}(\mathbf{x})\,\mathbf{g}(\mathbf{x})\,,\, -\mathbf{H}^{-1}(\mathbf{x})) \tag{6}$$

Note that Equation 5 is simply the full Newton step (Nocedal and Wright 2006a). We can therefore think of SNS as the stochastic counterpart of Newton-Raphson (NR) optimization. In NR optimization, we select the mean of the fitted Gaussian as the next step, while in SNS we draw a sample from the fitted Gaussian and apply MH test to accept or reject it. Also, note that in the special case where the sampled PDF is Gaussian, $f(\mathbf{x})$ is quadratic and

therefore the proposal function is identical to the sampled PDF. In this case $A(\mathbf{z}', \mathbf{z})$ is always equal to 1, implying an acceptance rate of 100%.

Contrary to standard Metropolis variants with multivariate Gaussians centered on current point (symmetric Gaussian proposal), SNS is an aggressive, non-local MCMC algorithm as it seeks to construct a global, Gaussian approximation of the PDF. Therein lies both its strength and its weakness. When the Gaussian approximation is sufficiently close to the true PDF, this can lead to a nearly-uncorrelated chain of samples, with the extreme case of perfectly uncorrelated samples for a multivariate Gaussian distribution. In such cases, SNS can be much more efficient than univariate and multivariate alternatives, including Metropolis sampling with symmetric Gaussian proposal. On the other hand, if the Gaussian approximation is poor, there can be little overlap between the two, resulting in low acceptance rate and inefficient sampling.

Our empirical observations indicate that, in exchangeable regression models SNS performs best when the number of observations is large compared to the number of variables. Appendix C provides theoretical support for this observation in the special case of a univariate log-density. The **sns** package contains the 'state space partitioning' strategy for improving mixing for SNS in high-dimensional problems (Section 3.2). Performance of SNS vis-a-vis other sampling techniques under various data regimes is further illustrated in Section 5.

2.3. Log-density concavity

Constructing a valid SNS proposal function (Eq. 6) requires the sampled log-density to be twice-differentiable and concave. Equivalently, the Hessian of log-density must exist and be negative definite. Many common probability distributions enjoy such property, perhaps with suitable transformations of parameters. We refer the reader to Table 2 of Gilks and Wild (1992) for a list of distributions and parameters that enjoy log-concavity. For Bayesian problems where log-density is a sum of contributions from log-prior and log-likelihood terms, it is sufficient for each term to be concave, since this property is additive.

An important application of the Bayesian framework is exchangeable regression models (Good 2002), where 1) log-likelihood is the sum of contributions from individual data points, and 2) one or more parameters of a probability distribution are assumed to be (possibly nonlinear functions of) the inner product of a vector covariates and a vector of coefficients, often referred to as 'linear predictors'. In such cases, log-density can be generically written in the following form:

$$L(\boldsymbol{\beta}^1, \ldots, \boldsymbol{\beta}^J) = \sum_{n=1}^{N} f_n(< \mathbf{x}_n^1, \boldsymbol{\beta}^1 >, \ldots, < \mathbf{x}_n^J, \boldsymbol{\beta}^J >), \tag{7}$$

where $< \boldsymbol{a}, \boldsymbol{b} >$ is the inner product of vectors $\boldsymbol{a}$ and $\boldsymbol{b}$, $\mathbf{x}_n^1, \ldots, \mathbf{x}_n^J$ are the covariate vectors $1, \ldots, J$ for the n'th observation, and $\boldsymbol{\beta}^1, \ldots, \boldsymbol{\beta}^J$ are the vectors of coefficients corresponding to J slots in the probability distributions $f_n(u^1, \ldots, u^J)$, $n = 1, \ldots, N$. Typically, J equals 1 or 2, with a notable exception being the multinomial distribution, used in multi-class logistic regression (Hasan, Zhiyu, and Mahani 2014), where J equals the number of classes.

A convenient property of the functional form in Eq. 7 is that the definiteness property of its Hessian as a function of the high-dimensional state space of $\boldsymbol{\beta}$:

$$\boldsymbol{\beta} \equiv (\boldsymbol{\beta}^{1,t}, \ldots, \boldsymbol{\beta}^{J,t})^t, \tag{8}$$

can be reduced to a much simpler problem, i.e. definiteness of the J-dimensional functions f_n is Eq. 7, for $n = 1, \ldots, N$. This notion is formally captured in the following theorem:

Theorem 1. *If all f_n's in Equation 7 have negative-definite Hessians AND if at least one of J matrices $\mathbf{X}^j \equiv (\mathbf{x}_1^j, \ldots, \mathbf{x}_N^j)^t$ is full rank, then $L(\boldsymbol{\beta}^1, \ldots, \boldsymbol{\beta}^J)$ also has a negative-definite Hessian.*

Proof is provided in Appendix B. Reasoning about definiteness of a matrix in a one- or two-dimensional space is much easier than doing so in a high-dimensional space, with dimensionality being the sum of lengths, K^j, of coefficients or $\sum_j K^j$. In a typical problem, this can sum be as large as 50 or 100. Theorem 1, on the other hand, means that for single-slot base distributions $f_n(u)$, we must simply prove that $f_n''(u) < 0$, $\forall n$. For multi-slot distributions, we can use Silvester's criterion for negative-definiteness, requiring that leading principal minors alternate between positive and negative (Gilbert 1991). In particular, for two-slot distributions, we require the diagonal elements to be negative while the determinant of the Hessian must be positive. It must be noted that Theorem 1 offers a sufficient, but not necessary, set of conditions. It is therefore possible for a log-density of the form shown in Eq. 7 to be negative-definite without satisfying the set of conditions stated in the above theorem. The log-density for heteroskedastic linear regression discussed in Section 4.4 is potentially one such example.

In some cases, while the Hessian cannot be proven to be negative-definite, yet blocks within the full Hessian have this property. In such cases, SNS can be combined with other, more generic sampling algorithms such as slice sampler or HMC, all embedded in a Gibbs cycle. The implementation would be similar to that of state space partitioning approach in **sns** (Section 3.2), but replacing SNS with alternative samplers for Hessian subsets (or blocks) that do not satisfy negative-definiteness. An example of this strategy, as well as the application of Theorem 1 is provided in in Section 4.4.

In addition to the theoeretical support provided in Theorem 1, **sns** also contains a utility function for empirical assessment of negative-definiteness for Hessian blocks. This feature is described in Section 3.5, and illustrated in Section 4.4.

2.4. Calculation of gradient and Hessian

Deriving the expressions for the gradient vector and the Hessian matrix, and implementing them as computer programs, can be tedious. For exchangeable regression models, we can apply the chain rule of derivatives to log-density of Eq. 7 to express the high-dimensional gradient and Hessian in terms of their low-dimensional counterparts for f_n's. The first derivative of log-likelihood can be written as:

$$\mathbf{G}(\boldsymbol{\beta}) \equiv \frac{\partial L}{\partial \boldsymbol{\beta}} = ((\frac{\partial L}{\partial \boldsymbol{\beta}^1})^t, \ldots, (\frac{\partial L}{\partial \boldsymbol{\beta}^J})^t)^t, \tag{9}$$

where

$$(\frac{\partial L}{\partial \boldsymbol{\beta}^j})^t \equiv (\frac{\partial L}{\partial \beta_1^j}, \ldots, \frac{\partial L}{\partial \beta_{K^j}^j}). \tag{10}$$

For second derivatives we have:

$$\mathbf{H}(\boldsymbol{\beta}) \equiv \frac{\partial^2 L}{\partial \boldsymbol{\beta}^2} = \left[\frac{\partial^2 L}{\partial \boldsymbol{\beta}^j \partial \boldsymbol{\beta}^{j'}}\right]_{j,j'=1,\ldots,J}, \tag{11}$$

where we have defined $\mathbf{H}(\boldsymbol{\beta})$ in terms of J^2 matrix blocks:

$$\frac{\partial^2 L}{\partial \boldsymbol{\beta}^j \partial \boldsymbol{\beta}^{j'}} \equiv \left[\frac{\partial L}{\partial \beta_k^j \partial \beta_{k'}^{j'}}\right]_{j=1,\ldots,K^j;j'=1,\ldots,K^{j'}} \tag{12}$$

Applying the chain rule to the log-likelihood function of Equation 7, we derive expressions for its first and second derivatives as a function of the derivatives of the base functions $f_1, \ldots, f_N$:

$$\frac{\partial L}{\partial \boldsymbol{\beta}^j} = \sum_{n=1}^{N} \frac{\partial f_n}{\partial \boldsymbol{\beta}^j} = \sum_{n=1}^{N} \frac{\partial f_n}{\partial u^j} \mathbf{x}_n^j = \mathbf{X}^{j,t} \mathbf{g}^j, \tag{13}$$

with

$$\mathbf{g}^j \equiv (\frac{\partial f_1}{\partial u^j}, \ldots, \frac{\partial f_N}{\partial u^j})^t, \tag{14}$$

and

$$\mathbf{X}^j \equiv (\mathbf{x}_1^j, \ldots, \mathbf{x}_N^j)^t. \tag{15}$$

Similarly, for the second derivative we have:

$$\frac{\partial^2 L}{\partial \boldsymbol{\beta}^j \partial \boldsymbol{\beta}^{j'}} = \sum_{n=1}^{N} \frac{\partial^2 f_n}{\partial \boldsymbol{\beta}^j \partial \boldsymbol{\beta}^{j'}} = \sum_{n=1}^{N} \frac{\partial^2 f_n}{\partial u^j \partial u^{j'}} (\mathbf{x}_n^j \otimes \mathbf{x}_n^{j'}) = \mathbf{X}^{j,t} \mathbf{h}^{jj'} \mathbf{X}^{j'}, \tag{16}$$

where $\mathbf{h}^{jj'}$ is a diagonal matrix of size N with n'th diagonal element defined as:

$$h_n^{jj'} \equiv \frac{\partial^2 f_n}{\partial u^j \partial u^{j'}} \tag{17}$$

We refer to the matrix form of the Equations 13 and 16 as 'compact' forms, and the explicit-sum forms as 'explicit' forms. The expander functions in the R package **RegressionFactory** (Mahani and Sharabiani 2015b) use the compact form to implement the high-dimensional gradient and Hessian, while proof of Theorem 1 utilizes the explicit-sum form of Equation 16 (see Appendix B).

In addition to the expander framework discussed above, users can also utilize the numerical differentiation facility of **sns** to circumvent the need for deriving analytical expressions for gradient and Hessian. This is explained in Section 3.5, and illustrated using an example in Section 4.4.

3. Software implementation and features

This section begins with an overview of functions included in **sns**. This is followed by a deep-dive into facilities for improving convergence and mixing of MCMC chains in SNS, diagnostic tools, and full Bayesian prediction. A discussion of capabilities for calculating and validatig log-density derivatives concludes this section.

,

3.1. Overview

The workhorse of **sns** package is the `sns` function, responsible for implementation of MH algorithm using the multivariate Gaussian proposal density described in Section 2.2. `sns` implements the following steps:

1. Evaluate the log-density function and its gradient and Hessian at $\mathbf{x}_{\text{old}}$: $f_{\text{old}}, \mathbf{g}_{\text{old}}, \mathbf{H}_{\text{old}}$.
2. Construct the multivariate Gaussian proposal function at $q(.|\mathbf{x}_{old})$ using Equation 6 and $\mathbf{x} = \mathbf{x}_{old}$.
3. Draw a sample $\mathbf{x}_{prop}$ from $q(.|\mathbf{x}_{old})$, and evaluate $logq_{prop} = \log(q(\mathbf{x}_{prop}|\mathbf{x}_{old}))$.
4. Evaluate the log-density function and its gradient and Hessian at $\mathbf{x}_{prop}$: $f_{prop}, \mathbf{g}_{prop}, \mathbf{H}_{prop}$.
5. Construct the multivariate Gaussian proposal function at $q(.|\mathbf{x}_{prop})$ using Equation 6 and $\mathbf{x} = \mathbf{x}_{prop}$, and evaluate $logq_{old} = \log(q(\mathbf{x}_{old}|\mathbf{x}_{prop}))$.
6. Calculate the ratio $r = \exp((f_{prop} - f_{old}) + (logq_{old} - logq_{prop}))$.
7. If $r \geq 1$ accept $\mathbf{x}_{prop}$: $\mathbf{x}_{new} \leftarrow \mathbf{x}_{prop}$. Else, draw a random deviate s from a uniform distribution over $[0, 1)$. If $s < r$, then accept $\mathbf{x}_{prop}$: $\mathbf{x}_{new} \leftarrow \mathbf{x}_{prop}$, else reject $\mathbf{x}_{prop}$: $\mathbf{x}_{new} \leftarrow \mathbf{x}_{old}$.

Fitting the multivariate Gaussian in steps 2 and 5 is done via calls to the private function `fitGaussian` in the **sns** package. We use the functions `dmvnorm` and `rmvnorm` from package **mvtnorm** to calculate the log-density of, and draw samples from, multivariate Gaussian proposal functions.

There are three important arguments in `sns`: `rnd`, `part`, and `numderiv`. `rnd` controls whether the algorithm should run in stochastic or MCMC mode (which is the default choice), or in non-stochastic or Newton-Raphson (NR) mode. `part` controls the state space partitioning strategy. These two arguments and their roles are described in Section 3.2. `numderiv` determines whether numerical differentiation must be used for calculating the gradient and/or Hessian of log-density, and is discussed in Section 3.5.

`sns.run` is a wrapper around `sns`, and offers the following functionalities:

1. Convenience of generating multiple samples via repeated calls to `sns`. After the first call, the Gaussian fit object (attached as attribute `gfit` in the returned value from `sns`) is fed back to `sns` via argument `gfit`, in order to avoid unnecessary fitting of the proposal function at current value.
2. Collecting diagnostic information such as log-probability time series, acceptance rate, relative deviation from quadratic approximation (time series), and components of MH test. These diagnostic measures are discussed in Section 3.3, and their use is illustrated via examples in Section 4.

Note that, when SNS is part of a Gibbs cycle and used in conjunction with other samplers, `sns.run` cannot be used since the conditional distribution that must be sampled from changes in each iteration.

The generic methods `summary.sns`, `plot.sns` and `predict.sns` provide diagnostic, visualization, and prediction capabilities. They are discussed in Sections 3.3 and 3.4, and illustrated via examples in Section 4.

3.2. Improving convergence and mixing

NR mode: Far from the distribution mode, the local multivariate Gaussian fit can be severely different from the PDF, leading to small overlap between the two, low acceptance rate and hence bad convergence. This can be overcome by spending the first few iterations in non-stochastic or NR mode, where instead of drawing from the proposal function we simply accept its mean as the next step. Rather than taking a full Newton step, we have implemented line search (Nocedal and Wright 2006b) to ensure convergence to the PDF maximum. To use `sns` in NR mode, users can set the argument `rnd` to `FALSE`. In NR mode, each iteration is guaranteed to increase the log-density. Using the NR mode during the initial burn-in phase is illustrated in Section 4. In `sns.run`, the argument `nnr` controls how many initial iterations will be performed in NR mode.

State space partitioning: Even when near the PDF maximum, the fitted Gaussian can be severely different from the PDF. This can happen if the PDF has a significant third derivative, a phenomenon that we have observed for high-dimensional problems, especially when the number of observations is small. To improve bad mixing in high dimensions, we use a strategy which we refer to as 'state space partitioning', where state space is partitioned into disjoint subsets and SNS is applied within each subset, wrapped in a Gibbs cycle. This functionality is available via the `part` argument, which is a list containing the set of state space dimensions belonging to each subset. Convenience functions `sns.make.part` and `sns.check.part` allow users to easily create partition lists and check their validity, respectively.

3.3. Diagnostics

sns includes a rich set of diagnostics which can be accessed via functions `summary.sns` and `plot.sns`. Some of these are generic measures applicable to all MCMC chains, some are specific to MH-based MCMC algorithms, and some are even further specialized for SNS as a particular flavor of MH. Where possible, we have used the library **coda** and adopted its naming conventions, but opted to create and maintain an independent set of functions due to their specialized and extended nature.

MCMC diagnostics: In `summary.sns`, we calculate the usual MCMC chain summaries including mean, standard deviation, quantiles, and effective sample size. We also calculate a sample-based p-value for each coordinate. In `plot.sns` we have log-density trace plot, state vector trace plots, effective sample size by coordinate, state vector histograms, and state vector autocorrelation plots.

MH diagnostics: In `summary.sns`, we calculate the acceptance rate of MH transition proposals. If `mh.diag` flag is set to `TRUE`, all 4 components of the MH test (`log.p`, `log.p.prop`, `log.q` and `log.q.prop`) for all iterations are returned as well.

SNS diagnostics: In `summary.sns`, we return `reldev.mean` (if `sns.run` was called with `mh.diag` set to `TRUE`), defined as the average relative deviation of log-density change (with respect to PDF maximum) from quadratic approximation (also constructed at PDF maximum). The location of PDF maximum is extracted from the Gaussian fit in the last iteration under NR mode. The higher this value, the more likely it is for the SNS to exhibit bad mixing. This is illustrated in Section 4. For `reldev.mean` to be valid, the user must ensure that the value of the argument `nnr` supplied to `sns.run` is sufficiently high to achieve convergence by the end of NR phase. Note that this is generally unfeasible if SNS is part of a Gibbs cycle, and used in conjunction with other samplers, as in this case the conditional distribution that SNS samples from constantly from one iteration to the next.

3.4. Full Bayesian prediction

The function `predict.sns` allows for full Bayesian prediction, using a sample-based representation of predictive posterior distribution. It accepts an arbitrary function of state vector as argument `fpred`, and applies the function across all samples of state vector, supplied in the first argument, which must be an output of `sns.run`. The core philosophy in full Bayesian prediction is to postpone summarization of samples until the last step. For example, rather than supplying the expected values of coefficients into a function, we supply the samples and take the expected value after applying the function. Following this proper approach is important because:

1. Mathematically, an arbitrary function is not commutable with the expected value operator. Therefore, applying expected value early produces incorrect results.

2. For a similar reason, confidence intervals cannot be propagated through arbitrary functions. Therefore, correct uncertainty measurement also requires a full Bayesian approach.

Note that `fpred` can be deterministic or stochastic. An example of the latter case is when we need to generate samples from the predictive posterior distribution, and `fpred` generates sample(s) from the conditional distribution $p(\hat{y}|\theta)$, where $\hat{y}$ is a new observation (for which we are seeking prediction) and θ represents the model parameters (can be a vector), for which have used SNS to generate samples. This can be considered a special case of ancestral sampling (Bishop 2006), applied to the following:

$$p(\hat{y}) = \int p(\hat{y} \,|\, \theta)\, p(\theta)\, \mathrm{d}\theta \tag{18}$$

3.5. Calculation and validation of log-density derivatives

SNS requires log-density to be twice-differentiable and concave. The **sns** package offers a numerical differentiation capability to circumvent the need to mathematically compute the log-density gradient and Hessian, and to help users determine whether the log-density (likely) satifies the twice-differentiability and concavity requirements.

The functions `grad` and `hessian` from **numDeriv** package (Gilbert and Varadhan 2012) are used for numerical differentiation. Arguments `numderiv`, `numderiv.method` and `numderiv.args` in the `sns` and `sns.run` functions control the usage and parameters of these functions. Users have three choices with regards to log-density derivatives:

1. `fghEval` returns a list with elements `f`, `g`, and `h` for log-density, its gradient and its Hessian, respectively. In this case, users must leave `numderiv` at the default of 0 (no numerical differentiation).

2. `fghEval` returns a list with elements `f` (log-density) and g (gradient), and Hessian is found through numerical differentiation by setting `numderiv` to 1.

3. `fghEval` returns the log-density only (no list), and both gradient and Hessian are calculated numerically by setting `numDeriv` to 2.

A slightly more efficient approach is to do 'numerical augmentation' of the log-density once, outside `sns`, via the utility function `sns.fghEval.numaug`. Section 4.4 contains an example of using numerical derivatives with SNS.

4. Using sns

In this section, we illustrate the core functionalities of **sns** as well as some of its advanced features via 4 examples. Below is an overview of objectives in each example:

1. Example 1: basic use of `sns.run` and `summary.sns`, ideal behavior of SNS for multivariate Gaussian PDFs
2. Example 2: using SNS in random and non-random modes, full Bayesian prediction of mean and actual response
3. Example 3: state space partitioning for high-dimensional PDFs
4. Example 4: log-concavity invariance theorem, expander framework for log-density derivatives, log-density validation, numerical differentiation

Setup details such as platform, OS, and R and package versions can be found in Appendix A. To maximize the reproducibility of results for each code segment, we repeatedly fix random seeds throughout the paper (using `set.seed()` function).

4.1. Example 1: Multivariate Gaussian

First, we launch an R session and load the **sns** package as well **mvtnorm** (used for evaluating the multivariate Gaussia log-density in `logdensity.mvg`). We also select a seed for rnadom number generator:

```
R> library("sns")
R> library("mvtnorm")
R> my.seed <- 0
```

Using **sns** to sample from a multivariate Gaussian is a contrived, but pedagogical, example. Since log-density for a multivariate Gaussian is quadratic, its second-order Taylor series expansion is not approximate but exact. In other words, the proposal function becomes location-independent, and equal to the sampled distribution. This means that 1) the MH test is always accepted, and 2) consecutive samples are completely independent, and hence the resulting chain is no longer Markovian. Of course, since we know how to sample from multivariate Gaussian proposal functions, we might as well directly sample from the original multivariate Gaussian distribution. (Hence, the pedagogical nature of this example.) To utilize **sns**, we must first implement the log-density and its gradient and Hessian:

```
R> logdensity.mvg <- function(x, mu, isigsq) {
+    f <- dmvnorm(x = as.numeric(x),
+      mean = mu, sigma = solve(isigsq), log = TRUE)
+    g <- - isigsq %*% (x - mu)
```

```
+    h <- -isigsq
+    return (list(f = f, g = g, h = h))
+  }
```

We now draw 500 samples from this log-desity, using pre-specified values for `mu` (mean vector) and `isigsq` (inverse of the covariance matrix, or precision matrix) in a 3-dimensional state space:

```
R> set.seed(my.seed)
R> K <- 3
R> mu <- runif(K, min = -0.5, max = +0.5)
R> isigsq <- matrix(runif(K*K, min = 0.1, max = 0.2), ncol = K)
R> isigsq <- 0.5*(isigsq + t(isigsq))
R> diag(isigsq) <- rep(0.5, K)
R> x.init <- rep(0.0, K)
R> x.smp <- sns.run(x.init, logdensity.mvg, niter = 500,
+    mh.diag = TRUE, mu = mu, isigsq = isigsq)
```

Next, we use the `summary.sns` function to view some of the diagnostics:

```
R> summary(x.smp)

Stochastic Newton Sampler (SNS)
state space dimensionality:  3
total iterations:  500
        NR iterations:  10
        burn-in iterations:  250
        end iteration:  500
        thinning interval:  1
        sampling iterations (before thinning):  250
acceptance rate:  1
        mean relative deviation from quadratic approx: 3.93e-15 % (post-burnin)
sample statistics:
        (nominal sample size: 250)
       mean        sd       ess      2.5%       50%  97.5% p-val
1   0.35400   1.65000 223.96358  -2.57369   0.32924 3.5844 0.880
2  -0.27201   1.64062 250.00000  -3.66272  -0.12906 2.6939 0.912
3  -0.10011   1.50216 250.00000  -3.14307  -0.10746 2.6271 0.944
summary of ess:
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  224.0   237.0   250.0   241.3   250.0   250.0
```

As expected, the acceptance rate is 100%, and there is no deviation from quadratic approximation, for SNS sampling of a multivariate Gaussian.

In real-world applications, the Gaussian proposal function is only an approximation for the sampled distribution (since log-density is not quadratic), creating the Markovian dependency and less-than-perfect acceptance rate. We study one such example next.

4.2. Example 2: Bayesian Poisson regression

Generalized Linear Models (GLMs) (Nelder and Baker 1972) are an important family of statistical models with applications such as risk analysis (Sobehart *et al.* 2000), public health (Sharabiani *et al.* 2011) and political science (Gelman and Hill 2007). GLMs can be extended to incorporate data sparseness and heterogeneity via the Hierarchical Bayesian framework (Rossi *et al.* 2005) or to account for repeated measurements and longitudinal data via Generalized Linear Mixed Model (McCulloch 2006). With properly-chosen link functions, many GLMs are known - or can be easily proven - to have globally-concave log-densities with negative-definite Hessian matrices (Gilks and Wild 1992). As such, GLMs are excellent candidates for SNS. Embedded in Bayesian frameworks, they continue to enjoy log-concavity assuming the same property holds for prior terms, according to the Bayes' rule and the invariance of concavity under function addition.

In our second example, we illustrate how to apply **sns** to the log-likelihood of Poisson regression. As before, we start with constructing the log-density and its gradient and Hessian, this time using the expander framework of **RegressionFactory** package(Mahani and Sharabiani 2015b):

```
R> loglike.poisson <- function(beta, X, y) {
+    regfac.expand.1par(beta, X = X, y = y,
+      fbase1 = fbase1.poisson.log)
+  }
```

Now we simulate data from the generative model:

```
R> set.seed(my.seed)
R> K <- 5
R> N <- 1000
R> X <- matrix(runif(N * K, -0.5, +0.5), ncol = K)
R> beta <- runif(K, -0.5, +0.5)
R> y <- rpois(N, exp(X %*% beta))
```

For reference, we do a maximum-likelihood (ML) estimation of the coefficients using `glm` command:

```
R> beta.init <- rep(0.0, K)
R> beta.glm <- glm(y ~ X - 1, family = "poisson",
+    start = beta.init)$coefficients
```

As mentioned before, **sns** can be run in non-stochastic mode, which is equivalent to Newton-Raphson optimization with line search. Results should be identical, or very close, to `glm` results:

```
     beta.glm    beta.nr
X1 -0.1893137 -0.1893137
X2 -0.4131500 -0.4131500
X3 -0.1598832 -0.1598832
X4 -0.1304481 -0.1304481
X5  0.3434516  0.3434516
```

```
R> beta.sns <- sns.run(beta.init, fghEval = loglike.poisson,
+    niter = 20, nnr = 20, X = X, y = y)
R> beta.nr <- beta.sns[20, ]
R> cbind(beta.glm, beta.nr)
```

The primary use-case for **sns** is not ML estimation, but rather MCMC sampling of the distribution. To do this, we perform the first few iterations in non-stochastic mode (20 iterations here), and then switch to stochastic mode for the remaining 180 iterations:

```
R> beta.smp <- sns.run(beta.init, loglike.poisson
+    , niter = 200, nnr = 20, mh.diag = TRUE, X = X, y = y)
```

Examining the log-density trace plot (Figure 1) shows an expected pattern: During non-stochastic phase (left of vertical line) log-probability rises steadily while approaching the peak. During MCMC sampling, on the other hand, PDF maximum forms an upper bound for the MCMC movements, and the chain occasionally visits low-probability areas. The plot is created using the following line:

```
R> plot(beta.smp, select = 1)
```

The `plot.sns` function offers 4 other types of plots, besides the log-density trace plot. See package documentation for details. Further diagnostic information can be accessed via the `summary.sns` function:

```
Stochastic Newton Sampler (SNS)
state space dimensionality:  5
total iterations:  200
        NR iterations:  20
        burn-in iterations:  100
        end iteration:  200
        thinning interval:  1
        sampling iterations (before thinning):  100
acceptance rate:  0.98
        mean relative deviation from quadratic approx: 0.282 % (post-burnin)
sample statistics:
        (nominal sample size: 100)
        mean         sd        ess       2.5%        50%   97.5% p-val
1  -0.191294   0.101788 100.000000  -0.399519  -0.183148 -0.0023  0.06 .
2  -0.416563   0.119709 100.000000  -0.648252  -0.426877 -0.1993  0.01 **
3  -0.152653   0.103663 100.000000  -0.340861  -0.148608  0.0497  0.14
4  -0.134873   0.107803  54.066600  -0.319464  -0.126089  0.0583  0.20
5   0.325324   0.089878 100.000000   0.176855   0.326557  0.5292  0.01 **
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
summary of ess:
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  54.07  100.00  100.00   90.81  100.00  100.00
```

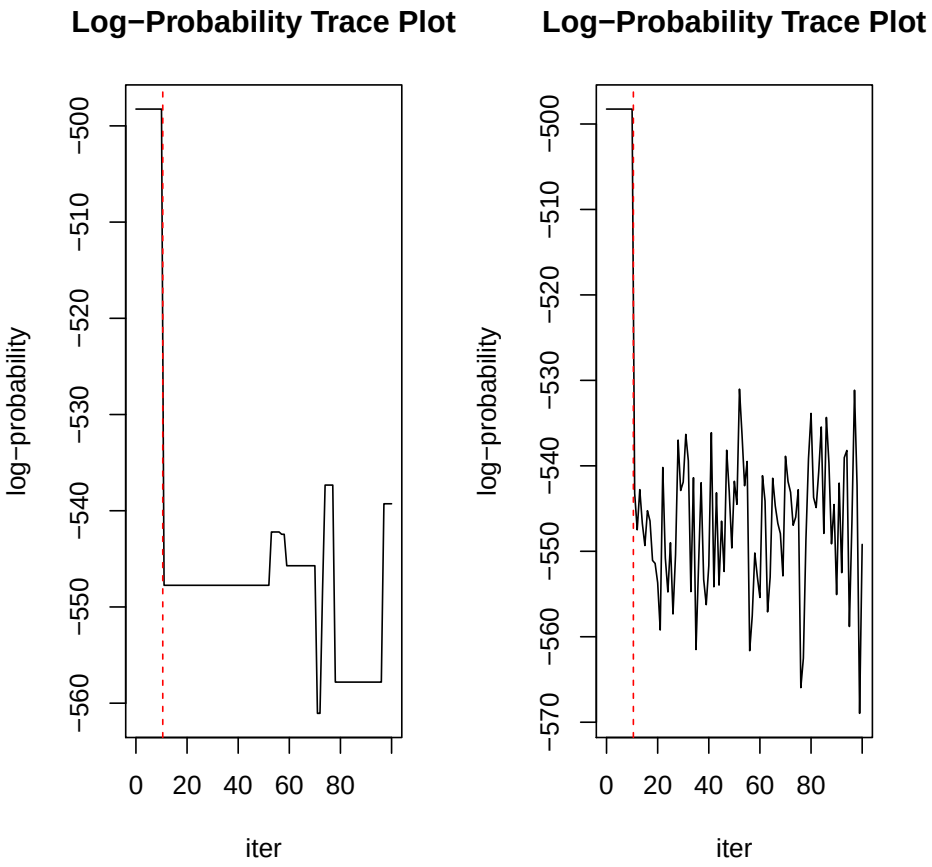

Figure 1: Log-density trace plot for Poisson regression problem with $K = 5$ and $N = 1000$. Vertical line separates non-stochastic mode (left, 20 iterations) from stochastic mode (right, 180 iterations).

```
R> summary(beta.smp)
```

The `summary.sns` function discards the first half of the samples as burn-in by default, before calculating sample statistics and acceptance. This behavior can be controlled via the argument `nburnin`. Arguments `end` and `thin` have behavior similar to their counterparts in the `as.mcmc` function of **coda** package. We observe two numbers in the summary print-out: Firstly, acceptance rate is less than 100%, contrary to the case with a multivariate Gaussian PDF (example 1). Secondly, the mean relative deviation from quadratic approximation (`reldev.mean`) is now non-zero, again reflecting non-Gaussianity of the poisson likelihood PDF. The number (<1%) is still small, however, leading to high acceptance rate and good mixing of the chain.

Next, we want to predict the response variable in Poisson regression model, given new values of the explanatory variables. We distinguish between two types of prediction: 1) predicting mean response, 2) generating samples from posterior predictive distribution. We illustrate how to do both using the `predict.sns` function. We begin by implementing the mean prediction function, which simply applies the inverse link function (exponential here) to the linear predictor. (For better comparison between the two prediction modes, we increase number of samples to 1000.)

```
R> beta.smp <- sns.run(beta.init, loglike.poisson,
+    niter = 1000, nnr = 20, mh.diag = TRUE, X = X, y = y)
R> predmean.poisson <- function(beta, Xnew) exp(Xnew %*% beta)
```

The following single line performs sample-based prediction of mean response (using `X` in lieu of `Xnew` for code brevity):

```
R> ymean.new <- predict(beta.smp, predmean.poisson,
+    nburnin = 100, Xnew = X)
```

`ymean.new` is a matrix of `N` (1000) rows and `niter - nburnin` (900) columns. Each row corresponds to an observation (one row of `Xnew`), and each column corresponds to a prediction sample (one row of `beta.smp` after burn-in).

We can also generate samples from posterior predictive distribution as follows:

```
R> predsmp.poisson <- function(beta, Xnew)
+    rpois(nrow(Xnew), exp(Xnew %*% beta))
R> ysmp.new <- predict(beta.smp, predsmp.poisson
+    , nburnin = 100, Xnew = X)
```

Comparing prediction summaries is illuminating:

```
prediction sample statistics:
        (nominal sample size: 900)
        mean         sd        ess       2.5%        50%  97.5%
1   1.097131   0.055727 745.264732   0.993232   1.096894 1.2046
2   0.843732   0.054143 715.905103   0.743759   0.839696 0.9601
3   0.890312   0.048949 739.896961   0.797980   0.888058 0.9929
```

```
4   0.870101   0.064002 900.000000   0.745485   0.871221 1.0004
5   0.788435   0.051432 900.000000   0.696831   0.784447 0.8981
6   1.452514   0.121211 814.557259   1.234916   1.441164 1.7091
...

R> summary(ymean.new)

prediction sample statistics:
        (nominal sample size: 900)
        mean         sd        ess       2.5%        50% 97.5%
1    1.06889    0.97678  900.00000    0.00000    1.00000     3
2    0.85222    0.92629  900.00000    0.00000    1.00000     3
3    0.88222    0.93475  900.00000    0.00000    1.00000     3
4    0.92444    0.94854 1020.54914    0.00000    1.00000     3
5    0.75778    0.87047  900.00000    0.00000    1.00000     3
6    1.48444    1.16530  900.00000    0.00000    1.00000     4
...

R> summary(ysmp.new)
```

In the limit of infinite samples, the mean predictions from the two methods will be equal, and they are quite close based on 900 samples above. However, standard deviation of predictions is much larger for `predsmp.poisson` compared to `predmean.poisson`, as the former combines the uncertainty of coefficient values (represented in the `sd` values for `beta`'s) with the uncertainty of samples from Poisson distribution around the mean, i.e., the `sd` of Poisson distribution. Also note that, as expected, quantiles for `predsmp.poisson` are discrete since predictions are discrete, while the quantiles for `predmean.poisson` are continuous as predictions are continuous in this case.

As mentioned in Section 2.2, there are two important cases where the Gaussian proposal function used in SNS deviates significantly from the underlying PDF, resulting in poor mixing of the MCMC chain. First, when sample size is small, and second when state space dimensionlity is high. The first scenario is addressed in performance benchmarking experiments of Section 5. Below we focus on SNS for high-dimensional PDFs.

4.3. Example 3: High-dimensional Bayesian Poisson regression

As discussed in Section 2.2, the multivariate Gaussian proposal in SNS tends to lose accuracy as state space dimensionality rises, leading to poor mixing of the resulting MCMC chain. Continuing with the Poisson regression example, we increase K from 5 to 100, while holding $N = 1000$. To illustrate that the problem is not covergence but mixing, we explicitly use the `glm` estimate (mode of PDF) as the initial value for the MCMC chain:

```
Stochastic Newton Sampler (SNS)
state space dimensionality:  100
total iterations:  100
        NR iterations:  10
        burn-in iterations:  50
```

```
        end iteration:  100
        thinning interval:  1
        sampling iterations (before thinning):  50
acceptance rate:  0.16
        mean relative deviation from quadratic approx: 5.25 % (post-burnin)
sample statistics:
        (nominal sample size: 50)
       mean        sd       ess      2.5%       50%   97.5% p-val
1 -0.459896  0.064151  5.362324 -0.621051 -0.434840 -0.4053  0.02 *
2  0.215889  0.056858 32.204465  0.129271  0.204078  0.4094  0.02 *
3  0.240960  0.118550 11.351804  0.011089  0.255061  0.3838  0.08 .
4 -0.093720  0.124676  5.034361 -0.400974 -0.081198  0.0237  0.76
5 -0.031323  0.080152  5.981971 -0.108520 -0.046393  0.1001  0.60
6  0.492257  0.083493 15.843840  0.260353  0.518507  0.6285  0.02 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
...
summary of ess:
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  1.596   3.956   6.346   8.480  12.410  32.204
```

```
R> set.seed(my.seed)
R> K <- 100
R> X <- matrix(runif(N * K, -0.5, +0.5), ncol = K)
R> beta <- runif(K, -0.5, +0.5)
R> y <- rpois(N, exp(X %*% beta))
R> beta.init <- glm(y ~ X - 1, family = "poisson")$coefficients
R> beta.smp <- sns.run(beta.init, loglike.poisson,
+    niter = 100, nnr = 10, mh.diag = TRUE, X = X, y = y)
R> summary(beta.smp)
```

We see a significant drop in acceptance rate as well as effective sample sizes for the coefficients. Also note that mean relative deviation from quadratic approximation is much larger than the value for $K = 5$. To improve mixing, we use the 'state space partitioning' strategy of **sns**, available via the `part` argument of `sns` and `sns.run`. This leads to SNS sampling of subsets of state space wrapped in Gibbs cycles, with each subset being potentially much lower-dimensional than the original, full space. This strategy can significantly improve mixing. Below we use the function `sns.make.part` to partition the 100-dimensional state space into 10 subsets, each 10-dimensional:

```
Stochastic Newton Sampler (SNS)
state space dimensionality:  100
state space partitioning:  10  subsets
total iterations:  100
        NR iterations:  10
        burn-in iterations:  50
        end iteration:  100
```

```
        thinning interval:  1
        sampling iterations (before thinning):  50
acceptance rate:  0.944
sample statistics:
        (nominal sample size: 50)
       mean        sd       ess      2.5%       50%   97.5% p-val
1 -0.577345  0.095907 50.000000 -0.755301 -0.567751 -0.4085  0.02 *
2  0.262993  0.100192 18.418195  0.072602  0.257516  0.4128  0.04 *
3  0.256804  0.095982 50.000000  0.072147  0.270832  0.4498  0.02 *
4 -0.143442  0.102042 21.673953 -0.327718 -0.155412  0.0474  0.28
5  0.048772  0.115663 19.773821 -0.161601  0.060174  0.2294  0.68
6  0.415373  0.093859 50.000000  0.227355  0.422012  0.5652  0.02 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
...
summary of ess:
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  8.049  24.524  50.000  41.667  50.000 215.828
```

```
R> beta.smp.part <- sns.run(beta.init, loglike.poisson,
+    niter = 100, nnr = 10, mh.diag = TRUE,
+    part = sns.make.part(K, 10), X = X, y = y)
R> summary(beta.smp.part)
```

Notice the improved acceptance rate as well as effective sample sizes. A comparison of log-probability trace plots confirms better mixing after convergence to PDF mode (see Figure 2).

```
R> par(mfrow = c(1,2))
R> plot(beta.smp, select = 1)
R> plot(beta.smp.part, select = 1)
```

4.4. Example 4: Heteroskedastic linear regression

For our last example, we study MCMC sampling of log-likelihood for heteroskedastic linear regression, where both mean and variance of probability distribution for response are dependent on a set of covariates. We cover several advanced topics in this examples: 1) application of invariance theorem (Section 2.3), 2) rapid construction of analytical functions for gradient and Hessian, 3) validation of log-density, 4) numerical calculation of derivatives, and 5) combining SNS with other samplers in Gibbs framework. The model can be formally stated as:

$$y_i \sim \mathcal{N}(\mathbf{x}_i^t \boldsymbol{\beta}\,,\, e^{\mathbf{z}_i^t \boldsymbol{\gamma}}), \tag{19}$$

where $\mathcal{N}(\mu, \sigma^2)$ is the normal distribution with mean μ and variance σ^2, $\mathbf{x}_i$ and $\mathbf{z}_i$ are vectors of covariates (explaining the mean and variance, respectively) for the i'th observation, y_i is the response variable for the i'th observation, and $\boldsymbol{\beta}$ and $\boldsymbol{\gamma}$ are vectors of coefficients for mean and variance. From Eq. 19, we can construct the following log-likelihood function:

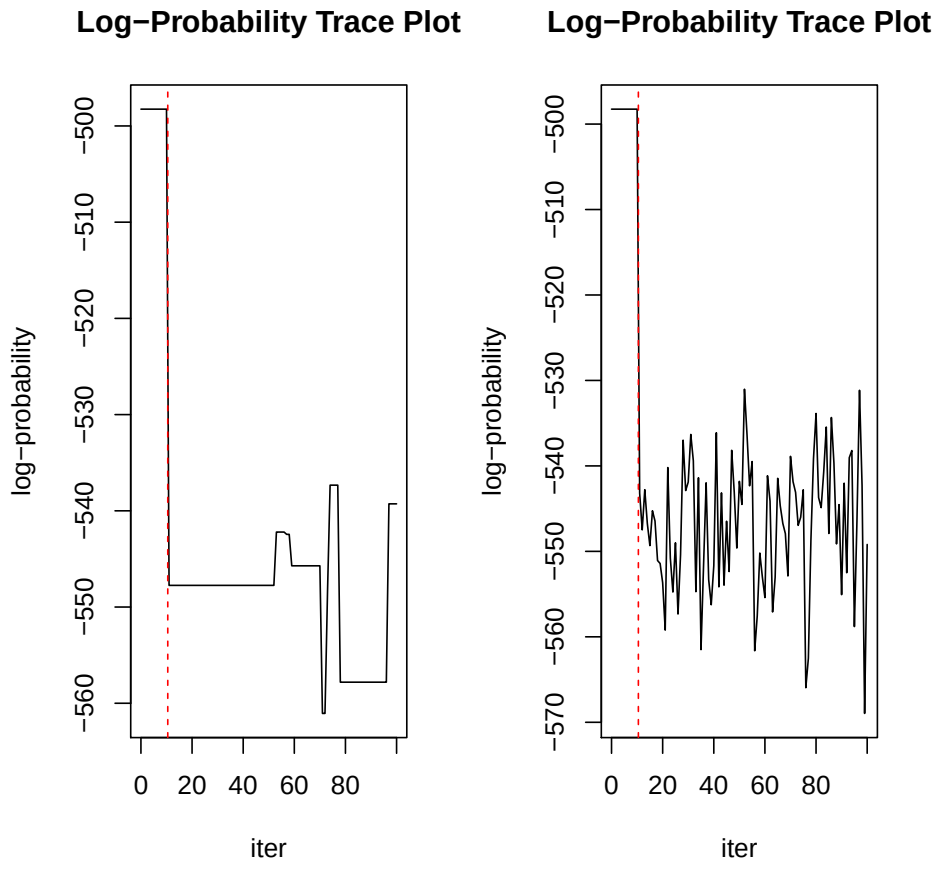

Figure 2: Comparison of log-density trace plots for Bayesian Poisson regression with $N = 1000$ and $K = 100$, without (left) and with (right) state space partitioning using 10 subsets.

```
R> loglike.linreg.het <- function(coeff, X, Z, y) {
+    K1 <- ncol(X)
+    K2 <- ncol(Z)
+    beta <- coeff[1:K1]
+    gamma <- coeff[K1 + 1:K2]
+
+    mu <- X %*% beta
+    sigma <- sqrt(exp(Z %*% gamma))
+    f <- sum(dnorm(y, mu, sigma, log = TRUE))
+
+    return (f)
+  }
```

Next, we simulate some data using the generative model:

```
R> set.seed(my.seed)
R> K1 <- 5
R> K2 <- 5
R> N <- 1000
R> X <- matrix(runif(N * K1, -0.5, +0.5), ncol = K1)
R> Z <- matrix(runif(N * K2, -0.5, +0.5), ncol = K2)
R> beta <- runif(K1, -0.5, +0.5)
R> gamma <- runif(K1, -0.5, +0.5)
R> mu <- X %*% beta
R> var <- exp(Z %*% gamma)
R> y <- rnorm(N, X %*% beta, sd = sqrt(var))
```

Before sampling from the log-density, we perform validation tests on it, using the utility function `sns.check.logdensity`. For `blocks` parameter, we focus on full Hessian as well as blocks corresponding to `beta` and `gamma`:

```
R> coeff.init <- rep(0.0, K1 + K2)
R> check.logdensity <- sns.check.logdensity(coeff.init, loglike.linreg.het
+    , X = X, Z = Z, y = y, dx = 1, nevals = 10
+    , blocks = list(1:(K1+K2), 1:K1, K1 + 1:K2))
R> check.logdensity

number of finite function evals: 10 (out of  10 )
recommended numderiv value: 2
is length of gradient vector correct? Yes
are dims of Hessian matrix correct? Yes
is numerical gradient finite? Yes
is numerical Hessian finite? Yes
is numeric Hessian (block) negative-definite? Yes Yes Yes
```

(In real applications, `nevals` must be increased to a number larger than 10.) The diagnostics output indicates that 1) log-density outputs are dimensionally valid, 2) function, gradient,

and Hessian are finite for all 100 points evaluated, and 3) log-density Hessian (and its specified blocks) is negative definite for all points evaluated.

Since the log-density for this problem falls under the 'exchangeable regression' category, the framework described in Section 2.3 can be applied for assessing concavity (Theorem 1) and constructing the derivatives (Eqn. 13 and 16). The log-density defined by `loglike.linreg.het` can be translated into the following expression for contribution of each data point towards log-density (ignoring constant terms):

$$f_i(\mu, \eta) = -\frac{\eta}{2} - \frac{(\mu - y_i)^2}{2} e^{-\eta}, \tag{20}$$

where μ and η are the linear predictors which assume values $\mathbf{x}_i^t \boldsymbol{\beta}$ and $\mathbf{z}_i^t \boldsymbol{\gamma}$, respectively, for observation i. According to Theorem 1, a sufficient condition for concavity of the log-density is that the Hessian of f_i's are negative-definite, for $i = 1, \ldots, N$, where N is the number of observations. Differentiating Eq. 20 readily leads to the following expression for the Hessian:

$$h_i = e^{-\eta} \begin{bmatrix} -1 & \mu - y_i \\ \mu - y_i & -\frac{(\mu - y_i)^2}{2} \end{bmatrix}. \tag{21}$$

While the diagonal elements of h_i are both negative, indicating that $\boldsymbol{\beta}$ and $\boldsymbol{\gamma}$ blocks are negative-definite, but the full Hessian cannot be proven to have the same property: According to Silvester's criterion for negative-definiteness, the second leading principal minor must be positive ($(-1)^2 = 1$). From Eq. 20, however, we see that

$$|h_i| = -\frac{(\mu - y_i)^2}{2} e^{-2\eta} \leq 0. \tag{22}$$

The above result does not contradict the output of `sns.check.logdensity`: On the one hand, Theorem 1 provides only a sufficient set of conditions. On the other hand, the emipirical checks performed by the software are limited to a finite set of points in the state space, and thus do not lead to any guarantees of negative definiteness everywhere.

While we can certainly try to sample from the full log-density using SNS, a safer option is to break down the full state space into subspaces corresponding to $\boldsymbol{\beta}$ and $\boldsymbol{\gamma}$, apply SNS to each subspace, and wrap these two steps in a Gibbs cycle. This will work because we have theoretical guarantee that the subspace Hessians are both negative-definite, as indicated by the diagonal elements of h_i's. To construct the derivatives for these two state subspaces, we can use the **RegressionFactory** expander framework in two ways: First, we can supply the base function `fbase2.gaussian.identity.log` to `regfac.expand.2par` and extract the subspace components from its output. This is easy to implement but is computationally inefficient since gradient and Hessian for the full state space are calculated during sampling from each subspace. Secondly, we can implement base functions for each subspace from scratch. For brevity of presentation, we opt for the first approach here:

```
R> loglike.linreg.het.beta <- function(beta, gamma, X, Z, y) {
+    K1 <- length(beta)
+    ret <- regfac.expand.2par(c(beta, gamma), X, Z, y
+      , fbase2 = fbase2.gaussian.identity.log)
+    return (list(f = ret$f, g = ret$g[1:K1], h = ret$h[1:K1, 1:K1]))
```

```
+  }
R> loglike.linreg.het.gamma <- function(gamma, beta, X, Z, y) {
+    K1 <- length(beta)
+    K2 <- length(gamma)
+    ret <- regfac.expand.2par(c(beta, gamma), X, Z, y
+      , fbase2 = fbase2.gaussian.identity.log)
+    return (list(f = ret$f, g = ret$g[K1 + 1:K2]
+              , h = ret$h[K1 + 1:K2, K1 + 1:K2]))
+  }
```

Having defined the subspace log-densities, we implement the Gibbs cycle:

```
           beta    beta.est
[1,]  0.1526770 -0.07868671
[2,] -0.4352875 -0.46271095
[3,]  0.1766124 -0.05479065
[4,]  0.2353717  0.18081163
[5,] -0.3887003 -0.35672998
          gamma  gamma.est
[1,] -0.4533454 -0.5362735
[2,] -0.3690897 -0.1911269
[3,]  0.3809564  0.4763015
[4,]  0.3397255  0.1722952
[5,]  0.3681427  0.2382708
```

```
R> nsmp <- 100
R> beta.iter <- rep(0.0, K1)
R> gamma.iter <- rep(0.0, K2)
R> beta.smp <- array(NA, dim = c(nsmp, K1))
R> gamma.smp <- array(NA, dim = c(nsmp, K1))
R> for (n in 1:nsmp) {
+    beta.iter <- sns(beta.iter, loglike.linreg.het.beta
+      , gamma = gamma.iter, X = X, Z = Z, y = y, rnd = nsmp>10)
+    gamma.iter <- sns(gamma.iter, loglike.linreg.het.gamma
+      , beta = beta.iter, X = X, Z = Z, y = y, rnd = nsmp>10)
+    beta.smp[n, ] <- beta.iter
+    gamma.smp [n, ] <- gamma.iter
+  }
R> beta.est <- colMeans(beta.smp[(nsmp/2+1):nsmp, ])
R> gamma.est <- colMeans(gamma.smp[(nsmp/2+1):nsmp, ])
R> cbind(beta, beta.est)
R> cbind(gamma, gamma.est)
```

In the above example, both $\boldsymbol{\beta}$ and $\boldsymbol{\gamma}$ subspaces are log-concave and thus we could use SNS for sampling from both inside the Gibbs cycle. In some problems, we may need to combine SNS with other samplers that do not require log-concavity. This is illustrated next, where we use slice sampler for $\boldsymbol{\gamma}$ subspace, using the **MfUSampler** package (Mahani and Sharabiani 2015a).

```
R> loglike.linreg.het.gamma.fonly <- function(gamma, beta, X, Z, y) {
+    return (regfac.expand.2par(c(beta, gamma), X, Z, y
+      , fbase2 = fbase2.gaussian.identity.log, fgh = 0))
+  }
R> beta.iter <- rep(0.0, K1)
R> gamma.iter <- rep(0.0, K2)
R> for (n in 1:nsmp) {
+    beta.iter <- sns(beta.iter, loglike.linreg.het.beta
+      , gamma = gamma.iter, X = X, Z = Z, y = y, rnd = nsmp>10)
+    gamma.iter <- MfU.Sample(gamma.iter, loglike.linreg.het.gamma.fonly
+      , beta = beta.iter, X = X, Z = Z, y = y)
+    beta.smp[n, ] <- beta.iter
+    gamma.smp [n, ] <- gamma.iter
+  }
R> beta.est.mix <- colMeans(beta.smp[(nsmp/2+1):nsmp, ])
R> gamma.est.mix <- colMeans(gamma.smp[(nsmp/2+1):nsmp, ])
R> cbind(beta, beta.est.mix)
R> cbind(gamma, gamma.est.mix)
```

Finally, we illustrate the use of numerical differentiation capabilities in `sns`, using the utility function `sns.check.logdensity`. We switch back to using SNS with γ subspace, but use `loglike.linreg.het.gamma.fonly`, and compute gradient and Hessian numerically, rather than analytically. Rather than requesting log-density augmentation with numerical derivatives to be performed once every iteration inside `sns`, we perform this once outside the loop, using `sns.fghEval.numaug`:

```
R> loglike.linreg.het.gamma.numaug <-
+    sns.fghEval.numaug(loglike.linreg.het.gamma.fonly, numderiv = 2)
R> beta.iter <- rep(0.0, K1)
R> gamma.iter <- rep(0.0, K2)
R> for (n in 1:nsmp) {
+    beta.iter <- sns(beta.iter, loglike.linreg.het.beta
+      , gamma = gamma.iter, X = X, Z = Z, y = y, rnd = nsmp>10)
+    gamma.iter <- sns(gamma.iter, loglike.linreg.het.gamma
+      , beta = beta.iter, X = X, Z = Z, y = y, rnd = nsmp>10)
+    beta.smp[n, ] <- beta.iter
+    gamma.smp [n, ] <- gamma.iter
+  }
R> beta.est.num <- colMeans(beta.smp[(nsmp/2+1):nsmp, ])
R> gamma.est.num <- colMeans(gamma.smp[(nsmp/2+1):nsmp, ])
R> cbind(beta, beta.est.num)
R> cbind(gamma, gamma.est.num)
```

5. Performance benchmarking

For log-densities with negative-definite Hessian, SNS can be an efficient MCMC algorithm. As mentioned in Section 2.3, many members of the exponential family belong in this category.

Family (link)		SNS	Slicer	ARS	AM	Speedup
Bernoulli (logit)	Time (sec)	14.4	90.8	281.4	19.5	
	ESS	7935	9401	9659	1551	
	Independent Samples / sec	553	103	34	80	5.4x
Poisson (log)	Time (sec)	13.9	73.1	264.6	15.4	
	ESS	6146	9523	9430	2514	
	Independent Samples / sec	444	130	36	164	2.7x
Exponential (log)	Time (sec)	13.9	73.6	273.1	15.4	
	ESS	5890	9650	9574	2364	
	Independent Samples / sec	423	131	35	154	2.7x

Table 1: Performance comparison of SNS against 3 other sampling techniques, using different probability distributions in a generalized linear model. In all cases, number of observations (N) is 1000, number of variables (K) is 10, and 10,000 samples were collected. SNS = stochastic Newton sampler; Slicer = univariate slice sampler with stepout and shrinkage; ARS = adaptive rejection sampling; AM = adaptive Metropolis. R package **MfUSampler** was used for Slicer and ARS, while **SamplerCompare** was used for AM. Effective sample size (ESS) was calculated using the 'Initial Sequence Estimators' method (Thompson 2010). All tuning parameters of samplers are at their default values in their respective packages.

In this section, we present some benchmarking results that help clarify the strengths and weaknesses of SNS vis-a-vis other sampling techniques, and thus provide empirical guidance for practitioners in choosing the right sampling algorithm for their problem.

In Table 1, we compare SNS against three alternative sampling algorithms: 1) univariate slice sampler with stepout and shrinkage (Neal 2003) or 'slicer', 2) adaptive rejection sampling (Gilks and Wild 1992) or 'ARS', which is another univariate sampler applicable to log-concave distributions, and 3) adaptive Metropolis (Roberts and Rosenthal 2009) or 'AM' using a multivariate Gaussian mixture proposal function. Performance is measured in 'independent samples per second', which is simply effective sample size divided by CPU time. We see that, for all three distributions studied - binomial, Poisson, and exponential - SNS achieves the best performance among all samplers, for this set of parameters: 1000 observations, 10 variables, and negligible correlation structure in the design matrix.

As discussed before, we expect SNS performance to deteriorate for small data sizes where deviation of sampled distribution from a Gaussian approximation is more significant. This point is illustrated in Figure 3 (left panels), where the performance of SNS, slicer, and AM are compared over a range of values for N ($K = 10$ and no correlation). We see that, while for $N > 200$, SNS has a clear advantage, slicer and AM tend to perform better for smaller data sets. Theoretical support for the univariate case is provided in Appendix C.

We had also pointed out that a weakness of univariate samplers such as slicer is when the sampled distribution exhibits strong correlations. This can be induced indirectly by increasing correlation in the matrix of covariates. As seen in Figure 3 (right panels), SNS performs rather steadily with increased correlation, while the performance of slicer and AM - particularly slicer - degrade as correlation increases.

Based on our benchmarking results, we recommend that SNS be considered as a primary MCMC sampling technique for log-concave distributions, moderate to large data, and partic-

ularly when significant correlation among covariates is suspected.

6. Discussion

In this paper we presented **sns**, an R package for Stochastic Newton Sampling of twice-differentiable, log-concave PDFs, where a multivariate Gaussian resulting from second-order Taylor series expansion of log-density is used as proposal function in a Metropolis-Hastings framework. Using an initial non-stochastic mode, equivalent to Newton-Raphson optimization with line search, allows the chain to rapidly converge to high-density areas, while 'state space partitioning', Gibbs sampling of full state space wrapping SNS sampling of lower-dimensional blocks, allows SNS to overcome mixing problems while sampling from high-dimensional PDFs. There are several opportunities for further research and development, which we discuss below.

Beyond twice-differentiability and log-concavity: Current version of SNS requires the log-density to be twice-differentiable and concave. In many real-world application, the posterior PDF does not have a negative-definite Hessian, or it cannot be proven to have such a property. In such cases, SNS would not be applicable to the entire PDF (but perhaps to subspaces, as illustrated in Example 4). A potential solution is discussed in Geweke and Tanizaki (2001) in the context of state-space models, where non-concave cases are handled by utilizing exponential and uniform proposal distributions. Convergence and mixing properties of such extensions to general cases must be carefully studied.

An important situation where twice-differentiability is violated is in the presence of boundary conditions. The current SNS algorithm assumes unconstrained state space. One way to deal with constrained subspaces is, again, to mix and match SNS and other samplers within Gibbs framework. For example, the slice sampler algorithm implemented in **MfUSampler** is capable of dealing with boxed constraints. It can therefore be assigned to subspaces with constraints, and the unconstrained subspaces can be handled by SNS. Further research is needed in order to relax twice-differentiability and log-concavity requirements for SNS.

Implementation Efficiency: The majority of time during typical applications of MCMC sampling is spent on evaluating the log-density and its derivatives. In particular, for Stochastic Newton Sampler, computing the Hessian matrix is often the most time-consuming step, which is $O(N \times K^2)$ where N is the number of observations and K is the number of attributes in a statistical model with i.i.d observations. In `sns` function, this step is performed by user-supplied function `fghEval`. On the other hand, the sampling algorithm itself is computationally dominated by $O(K^3)$ steps such as Cholesky factorization of the Hessian (inside `fitGaussian` or `rmvnorm`). As long as K is much smaller than N, therefore, the end-to-end process is dominated by `fghEval`. A notable exception will be applications with 'wide data' such as those encountered in DNA microarray analysis. Therefore, the best way to improve the efficiency of SNS is for the user to supply a high-performance `fghEval` function to `sns`, e.g., through compiled code, parallelization, etc.

Computational implications of subset size during state space partitioning require more attention as well. In particular, the current implementation in **sns** calls the same underlying, full-space `fghEval` function. This means within each subset, full Hessian is calculated, which is computationally wasteful. This naive approach neutralizes a potential computational advantage of state space partitioning, i.e., reducing the cost of Hessian evaluation which goes up quadratically with state space dimensionality. A more sophisticated approach would require

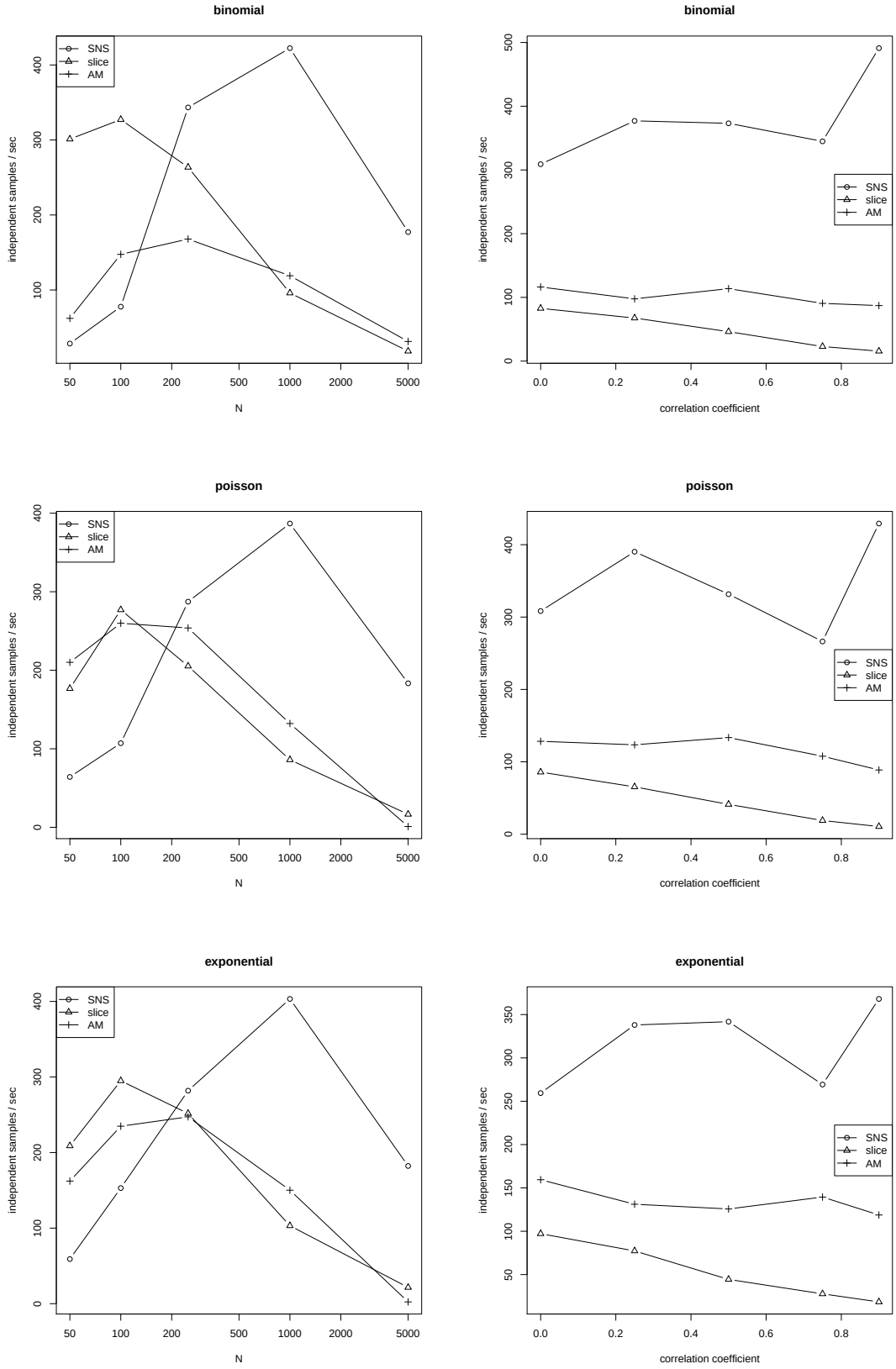

Figure 3: Impact of data size (left panels) and covariate correlation (right panels) on performance of SNS, slicer, and AM for binomial, Poisson, and exponential regression log-densities. Covariate vectors, forming rows of the covariate matrix (`X`), are sampled from multivariate Gaussian dsitribution with .

the user to supply a function and derivative evaluation routine that produces the derivatives over state subspaces. Here, the tradeoff is between computational efficiency vs. a more complex API for function evaluation, whose implementation for each problem would be the responsibility of the user.

Also, more comprehensive analysis - theoretical and empirical - is needed to determine when/how to partition the state space. The correlation structure of the state space might need to be taken into account while assigning coordinates to subsets, as well as data dimensions such as number of observations. Our preliminary, uni-dimensional analysis shows that mixing improves with $N^{1/2}$, where N is the number of independently-sampled observations in a regression problem.

A. Setup

Below is the **R** session information used for this entire paper. Note that the **sns** package used was built under R version 3.3.0.

```
R> sessionInfo()

R version 4.0.2 (2020-06-22)
Platform: x86_64-koji-linux-gnu (64-bit)
Running under: Amazon Linux 2

Matrix products: default
BLAS/LAPACK: /usr/lib64/R/lib/libRblas.so

locale:
 [1] LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C
 [3] LC_TIME=en_US.UTF-8        LC_COLLATE=C
 [5] LC_MONETARY=en_US.UTF-8    LC_MESSAGES=en_US.UTF-8
 [7] LC_PAPER=en_US.UTF-8       LC_NAME=C
 [9] LC_ADDRESS=C               LC_TELEPHONE=C
[11] LC_MEASUREMENT=en_US.UTF-8 LC_IDENTIFICATION=C

attached base packages:
[1] stats     graphics  grDevices utils     datasets  methods   base

other attached packages:
[1] RegressionFactory_0.7.4 mvtnorm_1.1-3           sns_1.2.2

loaded via a namespace (and not attached):
[1] compiler_4.0.2      tools_4.0.2         coda_0.19-4
[4] grid_4.0.2          numDeriv_2016.8-1.1 lattice_0.20-41
```

B. Invariance of Hessian definiteness under linear transformations

To prove negative-definiteness of $\mathbf{H}(\boldsymbol{\beta})$ (hereafter referred to as $\mathbf{H}$ for brevity), we seek to prove that $\mathbf{p}^t\mathbf{H}\mathbf{p}$ is negative for all non-zero $\mathbf{p}$ in $\mathbb{R}^{\sum_j K^j}$. We begin by decomposing $\mathbf{p}$ into J subvectors of length K^j each:

$$\mathbf{p} = (\mathbf{p}^{1,t}, \ldots, \mathbf{p}^{J,t})^t. \tag{23}$$

We now have:

$$\begin{aligned}
\mathbf{p}^t\mathbf{H}\mathbf{p} &= \sum_{j,j'=1}^{J} \mathbf{p}^{j,t} \frac{\partial^2 L}{\partial \boldsymbol{\beta}^j \partial \boldsymbol{\beta}^{j'}} \mathbf{p}^{j'} && (24)\\
&= \sum_{j,j'} \mathbf{p}^{j,t} \left(\sum_n \frac{\partial^2 f_n}{\partial u^j \partial u^{j'}} \cdot (\mathbf{x}_n^j \otimes \mathbf{x}_n^{j'}) \right) \mathbf{p}^{j'} && (25)\\
&= \sum_n \sum_{j,j'} \frac{\partial^2 f_n}{\partial u^j \partial u^{j'}} \mathbf{p}^{j,t} (\mathbf{x}_n^j \otimes \mathbf{x}_n^{j'}) \mathbf{p}^{j'} && (26)
\end{aligned}$$

If we define a set of new vectors $\mathbf{q}_n$ as:

$$\mathbf{q}_n \equiv \left[\mathbf{p}^{1,t}\mathbf{x}_n^1 \quad \cdots \quad \mathbf{p}^{J,t}\mathbf{x}_n^J \right], \tag{27}$$

and use $\mathbf{h}_n$ to denote the J-by-J Hessian of f_n:

$$\mathbf{h}_n \equiv [h_n^{jj'}]_{j,j'=1,\ldots,J}, \tag{28}$$

we can write:

$$\mathbf{p}^t\mathbf{H}\mathbf{p} = \sum_n \mathbf{q}_n^t \, \mathbf{h}_n \, \mathbf{q}_n. \tag{29}$$

Since all $\mathbf{h}_n$'s are assumed to be negative definite, all $\mathbf{q}_n^t \, \mathbf{h}_n \, \mathbf{q}_n$ terms must be non-positive. Therefore, $\mathbf{p}^t\mathbf{H}\mathbf{p}$ can be non-negative only if all its terms are zero, which is possible only if all $\mathbf{q}_n$'s are zero vectors. This, in turn, means we must have $\mathbf{p}^{j,t}\mathbf{x}_n^j = 0, \ \forall\, n, j$. In other words, we must have $\mathbf{X}^j\mathbf{p}^j = \emptyset, \ \forall\, j$. This means that all $\mathbf{X}^j$'s have non-singleton nullspaces and therefore cannot be full-rank, which contradicts our assumption. Therefore, $\mathbf{p}^T\mathbf{H}\mathbf{p}$ must be negative. This proves that $\mathbf{H}$ is negative definite.

C. SNS mixing and number of observations

Efficient mixing in SNS depends on absence of large changes to the mean and std of the proposal function (relative to the std calculated at mode) as the chain moves within a few std's of the mode. To quantify this, we begin with the Taylor series expansion of log-density around its mode μ_0 (where $f'(\mu_0) = 0$), this time keeping the third-order term as well (and restricting our analysis to univariate case):

$$\begin{aligned}
f(x) &\approx f(\mu_0) - \frac{1}{2}\tau_0(x-\mu_0)^2 + \frac{1}{6}\kappa_0(x-\mu_0)^3, && (30)\\
\tau_0 &\equiv -f''(\mu_0) && (31)\\
\kappa_0 &\equiv f'''(\mu_0) && (32)
\end{aligned}$$

Applying the Newton step to the above formula, we can arrive at mean $\mu(x)$ and precision $\tau(x)$ of the fitted Gaussian at x:

$$\begin{aligned} \mu(x) &= x - \frac{f'(x)}{f''(x)} && \text{(33a)} \\ &= x - \frac{-\tau_0(x-\mu_0) + \frac{1}{2}\kappa(x-\mu_0)^2}{-\tau_0 + \kappa_0(x-\mu_0)} && \text{(33b)} \\ \tau(x) &= \tau_0 + \kappa_0(x-\mu_0) && \text{(33c)} \end{aligned}$$

We now form the following dimensionless ratios, which we need to be much smaller than 1 in order to have good mixing for MH-MGT:

$$|\mu(x) - \mu_0| \,.\, \tau_0 \ll 1 \quad (34)$$
$$|\tau(x) - \tau_0|/\tau_0 \ll 1 \quad (35)$$

when $|x - \mu_0| \,.\, \tau_0 \sim O(1)$. Some algebra shows that both these conditions are equivalent to:

$$\eta_0 \equiv |\kappa_0| \,.\, \tau_0^{-3/2} \ll 1 \quad (36)$$

In real-world applications, we are interested in efficient sampling from computationally-expensive log-densities. In particular, statistical models often involve many independent observations, each contributing an additive term to the log-likelihood function:

$$f(x) = \sum_{i=1}^{N} f_i(x) \quad (37)$$

We can therefore re-write equation (36) as:

$$\eta_0 = |\sum_{i=1}^{N} f_i'''(x)| \,.\, [\sum_{i=1}^{N} f_i''(x)]^{-3/2} \quad (38)$$

Assuming that individual terms in above equation remain bounded, we can easily see that

$$\eta_0 \sim O(N^{-1/2}) \quad (39)$$

We therefore arrive at the following rule of thumb: *SNS becomes more efficient as the number of observations in a log-likelihood function increases.*

References

Bishop CM (2006). *Pattern recognition and machine learning*. springer.

Duane S, *et al.* (1987). "Hybrid Monte Carlo." *Physics letters B*, **195**(2), 216–222.

Gamerman D (1997). "Sampling from the Posterior Distribution in Generalized Linear Mixed Models." *Statistics and Computing*, **7**(1), 57–68.

Gelman A, Hill J (2007). *Data Analysis Using Regression and Multilevel/Hierarchical Models*. Cambridge University Press.

Geman S, Geman D (1984). "Stochastic Relaxation, Gibbs Distributions, and the Bayesian Restoration of Images." *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, (6), 721–741.

Geweke J, Tanizaki H (2001). "Bayesian Estimation of State-Space Models Using the Metropolis–Hastings Algorithm Within Gibbs Sampling." *Computational Statistics & Data Analysis*, **37**(2), 151–170.

Gilbert GT (1991). "Positive definite matrices and Sylvester's criterion." *American Mathematical Monthly*, pp. 44–46.

Gilbert P, Varadhan R (2012). *numDeriv: Accurate Numerical Derivatives*. R package version 2012.9-1, URL http://CRAN.R-project.org/package=numDeriv.

Gilks WR, Wild P (1992). "Adaptive Rejection Sampling for Gibbs Sampling." *Applied Statistics*, pp. 337–348.

Girolami M, Calderhead B (2011). "Riemann Manifold Langevin and Hamiltonian Monte Carlo Methods." *Journal of the Royal Statistical Society B (Statistical Methodology)*, **73**(2), 123–214.

Good PI (2002). "Extensions of the concept of exchangeability and their applications." *Journal of Modern Applied Statistical Methods*, **1**(2), 34.

Hasan A, Zhiyu W, Mahani AS (2014). *mnlogit: Multinomial Logit Model*. R package version 1.2.0, URL http://CRAN.R-project.org/package=mnlogit.

Hastings WK (1970). "Monte Carlo Sampling Methods Using Markov Chains and Their Applications." *Biometrika*, **57**(1), 97–109.

Hoffman MD, Gelman A (2014). "The No-U-Turn Sampler: Adaptively Setting Path Lengths in Hamiltonian Monte Carlo." *The Journal of Machine Learning Research*, **15**(1), 1593–1623.

Mahani AS, Sharabiani MT (2015a). *MfUSampler: Multivariate-from-Univariate (MfU) MCMC Sampler*. R package version 0.9.2.

Mahani AS, Sharabiani MT (2015b). *RegressionFactory: Expander Functions for Generating Full Gradient and Hessian from Single- and Multi-Slot Base Distributions*. R package version 0.7.1, URL http://CRAN.R-project.org/package=RegressionFactory.

MATLAB (2014). *Release 2014b*. The MathWorks Inc., Natick, Massachusetts.

McCulloch CE (2006). *Generalized Linear Mixed Models*. John Wiley & Sons Online Library.

Neal RM (2003). "Slice Sampling." *The Annals of Statistics*, pp. 705–741.

Nelder JA, Baker R (1972). *Generalized Linear Models*. John Wiley & Sons Online Library.

Nocedal J, Wright SJ (2006a). *Numerical Optimization*. Springer Series in Operations Research and Financial Engineering. Springer-Verlag.

Nocedal J, Wright SJ (2006b). *Numerical Optimization*, chapter 3. Springer Series in Operations Research and Financial Engineering. Springer-Verlag.

Plummer M (2013). *JAGS: Just Another Gibbs Sampler, Version 3.4.0*. URL http://mcmc-jags.sourceforge.net/.

Qi Y, Minka TP (2002). "Hessian-Based Markov Chain Monte-Carlo Algorithms." *1st Cape Cod Workshop on Monte Carlo Methods*.

Roberts GO, Rosenthal JS (1999). "Convergence of Slice Sampler Markov Chains." *Journal of the Royal Statistical Society B (Statistical Methodology)*, **61**(3), 643–660.

Roberts GO, Rosenthal JS (2009). "Examples of adaptive MCMC." *Journal of Computational and Graphical Statistics*, **18**(2), 349–367.

Rossi PE, *et al.* (2005). *Bayesian Statistics and Marketing*. John Wiley & Sons.

Sharabiani MTA, *et al.* (2011). "Immunologic Profile of Excessive Body Weight." *Biomarkers*, **16**(3), 243–251.

Sobehart JR, *et al.* (2000). "Moody's Public Firm Risk Model: A Hybrid Approach to Modeling Short-Term Default Risk." *Moody's Investors Service, Global Credit Research, Rating Methodology, March*.

Stan Development Team (2014). "Stan: A C++ Library for Probability and Sampling, Version 2.5.0." URL http://mc-stan.org/.

Thomas A, O'Hara B, Ligges U, Sturtz S (2006). "Making BUGS Open." *R News*, **6**(1), 12–17.

Thompson MB (2010). "A comparison of methods for computing autocorrelation time." *arXiv preprint arXiv:1011.0175*.

Affiliation:

Alireza S. Mahani
Scientific Computing Group
Sentrana Inc.
1725 I St NW
Washington, DC 20006
E-mail: alireza.mahani@sentrana.com