National University of Singapore

M.Sc. Thesis for Physics

Hamiltonian Monte Carlo sampling from quantum state space

Author:
Boyu Wang

Supervisor:
Berthold-Georg Englert

A thesis submitted in partial fulfilment of the requirements for the degree of Master of Science

in the

Berge Englert Group
Department of Physics

May 2015

NATIONAL UNIVERSITY OF SINGAPORE

Abstract

Faculty of Science
Department of Physics

Master of Science

Hamiltonian Monte Carlo sampling from quantum state space

by Boyu WANG

Hamiltonian Monte Carlo (HMC), one typical method of Markov Chain Monte Carlo (MCMC) random walks, is an efficient way for sampling from quantum state space, due to its high and controllable acceptance rate, as well as its not strongly correlated sample points. Under suitable parameterizations of the density matrix and proper programming implementation, we are able to simulate and obtain samples with considerate amount of data, with respect to various prior probabilities. Properties of the samples are further analyzed, such as the probability distributions of purity, fidelity, distance, *etc.*

Acknowledgements

I wish to express my sincere thanks to Prof. Berthold-Georg Englert and Asst. Prof. Ng Hui Khoon for suggesting this project topic and bringing me into the research group. I also thank for the academic support, guidance and advice given by them during the project period.

I am grateful for the valuable suggestions and generous help from Dr. Shang Jiangwei. His encouragement and advice always guided me into the right direction.

I would also like to thank everyone in the research group, especially Hu Yuxin, Len Yink Loong, Li Xikun, Max Seah Yi-Lin, and Ye Luyao. It has been a wonderful experience working with them all.

Contents

List of Figures

Abbreviations

HMC	**H**amiltonian **M**onte **C**arlo
MCMC	**M**arkov **C**hain **M**onte **C**arlo
MLE	**M**aximum **L**ikelihood **E**stimator
POM	**P**robability **O**perator **M**easurement
SIC	**S**ymmetric **I**nformationally **C**omplete

To my grandparents

Chapter 1

Introduction

1.1 Point Likelihood

Quantum state tomography is the attempt of reconstructing a quantum state, by performing measurements on quantum systems described by identical density matrix ρ. Ideally, the measurements should be informationally complete, *i.e.*, the measurement operators form an operator basis of the Hilbert space of the system being measured.
Generally, for a set of probability-operator measurement (POM) of K outcomes, the positive operators $\{\Pi_i\}$ sum up to unity,

$$\sum_{i=1}^{K} \Pi_i = \mathbb{1} \tag{1.1}$$

Identical copies of a state ρ are measured repeatedly by this POM, resulting in probabilities of detector clicks

$$p_i = \mathrm{Tr}\{\rho\,\Pi_i\} = \langle \Pi_i \rangle \tag{1.2}$$

for the i-th detector, and they satisfy the following properties,

$$p_i \geq 0, \quad \sum_{i=1}^{K} p_i = 1 \tag{1.3}$$

After measuring N identical copies of the state ρ, we obtain measurement data D, formed by a sequence of detector clicks $\{n_1, n_2, \ldots, n_K\}$ with

$$\sum_{i=1}^{K} n_i = N \tag{1.4}$$

where n_i is the number of clicks for the i-th detector.
The probability of obtaining data D for a given state ρ is given by the point likelihood

$$L(D|\rho) = p_1^{n_1} p_2^{n_2} \ldots p_K^{n_K} = \prod_{i=1}^{K} p_i^{n_i} \tag{1.5}$$

The maximum-likelihood estimator (MLE) for state ρ is thereafter computed by maximizing the above equation. However, this point estimator is of limited power to make statistical inferences, as we will not be able to tell if the probability distribution is sharply peaked at MLE, or widely spreads out over the entire region. In order to answer such questions, we need to find a way to construct and characterize error bars and error regions.

1.2 Estimator Regions

1.2.1 Prior density

Before any measurement data D, our prior knowledge about the quantum system is called the prior probability. The size of an infinitesimal vicinity of state ρ, denoted by $(\mathrm{d}\rho)$, is

$$(\mathrm{d}\rho) = w(p)(\mathrm{d}p) \tag{1.6}$$

where $w(p)$ is the prior density, and $(\mathrm{d}p) = \mathrm{d}p_1 \mathrm{d}p_1 \cdots \mathrm{d}p_K$ is the infinitesimal volume in the probability space defined by the set of probabilities $\{p_1, p_2, \ldots, p_K\}$. The size[1] of a region $\mathcal{R}$ is defined as the probability that state ρ lies within the region and denoted by $S_{\mathcal{R}}$,

$$S_{\mathcal{R}} = \int_{\mathcal{R}} (\mathrm{d}\rho) = \int_{\mathcal{R}} w(p)(\mathrm{d}p) \tag{1.7}$$

with

$$S_{\mathcal{R}_0} = \int_{\mathcal{R}_0} (\mathrm{d}\rho) = 1 \tag{1.8}$$

for the entire state space $\mathcal{R}_0$.

1.2.2 Posterior density

The posterior density of state ρ is the conditional probability after measurement data D is taken into account, which is

$$P(\rho|D) = \frac{L(D|\rho)}{L(D)} \tag{1.9}$$

where $L(D)$ is the prior likelihood of measurement data D, by integrating $L(D|\rho)$ over the entire space $\mathcal{R}_0$, *i.e.*,

$$L(D) = \int_{\mathcal{R}_0} L(D|\rho)(\mathrm{d}\rho) = \int_{\mathcal{R}_0} L(D|p)w(p)(\mathrm{d}p) \tag{1.10}$$

We further define the credibility[1] of region $\mathcal{R}$, denoted by $c_{\mathcal{R}}$, to be the posterior probability of state ρ lies within the region, given measurement data D,

$$c_{\mathcal{R}} = \frac{\int_{\mathcal{R}} L(D|\rho)(\mathrm{d}\rho)}{\int_{\mathcal{R}_0} L(D|\rho)(\mathrm{d}\rho)} = \frac{1}{L(D)} \int_{\mathcal{R}} L(D|p)w(p)(\mathrm{d}p) \tag{1.11}$$

1.3 Monte Carlo Integration

1.3.1 Motivation

In order to compute $S_{\mathcal{R}}$ and $c_{\mathcal{R}}$ in Equations 1.7 and 1.11, a K-dimension integral is envolved, since the infinitesimal volume in probability space is given by

$$(\mathrm{d}p) = \mathrm{d}p_1 \mathrm{d}p_1 \cdots \mathrm{d}p_K \tag{1.12}$$

As we shall see later in Chapter 3, for a d-dimensional density matrix ρ, there are $(d^2 - 1)$ independent variables. Upon a re-parameterization of state ρ from probability space to a certain parameterization space, the integral is in fact of dimension $(d^2 - 1)$, *i.e.*,

$$(\mathrm{d}\rho) \propto (\mathrm{d}p) \propto (\mathrm{d}\theta), \text{ where } (\mathrm{d}\theta) = \mathrm{d}\theta_1 \mathrm{d}\theta_2 \cdots \mathrm{d}\theta_{d^2-1} \tag{1.13}$$

For instance, in the 2-qubit case, density operator ρ is a 4×4 matrix and the integral would become 15-dimensional. The dimensionality rapidly increases to 63 for a 3-qubit state.

Hence we employ the idea of Monte Carlo integration, where random sample points are generated based on our probability distribution $w(p)$, or in other words, $w(\theta)$. The values of integrand are then evaluated at these sample points, and finally the

integral is estimated numerically by a weighted sum over these values.
The sampling strategy is, however, not unique. In this work, we use Hamiltonian Monte Carlo (HMC) algorithm, which is a typical Markov-chain Monte Carlo (MCMC) method.[1] In MCMC, sample points are generated based on a Markov chain random walk, with respect to certain desired probability distribution. The quality of the sample depends on the number of sample points, instead of the dimensionality.
HMC algorithm is introduced in the next section. To implement HMC on quantum systems, we also need a suitable parameterization method for density matrix ρ, in terms of its (d^2-1) independent variables. We demonstrate two possible ways of parameterization in Chapter 2. It is therefore possible to obtain samples with, typically, 1 million data points within reasonable amount of time[2], subject to any prior or posterior density. In Chapter 3, we show some of the results and discuss properties of the samples. The MATLAB® code used is attached in Appendix A at the end of this report.

1.3.2 Hamiltonian Monte Carlo

In HMC, parameters $\{\theta_i\}$ are treated as position variables of an artificial system evolving under Hamiltonian dynamics, where the Hamiltonian H is assumed to be

$$H(\theta,\Theta)=\frac{\Theta^2}{2}+U(\theta), \tag{1.14}$$

where $U(\theta)$ is the potential energy given by $U(\theta)=-\log w(\theta)$, and $\{\Theta_i\}$ are the associated canonical momentum variables. Here $w(\theta)$ is our target density distribution.
More specifically, this Hamiltonian evolution is computed by the leapfrog method[3], with the following procedure:

Step 1. Set $i=1$, with initial condition $\{\theta(i-1),\Theta(i-1)\}$, and number-of-steps L, time sub-interval $\varepsilon=T/L$.

Step 2. Compute $\Theta(i-\frac{1}{2})=\Theta(i-1)-\frac{\varepsilon}{2}\nabla U(\theta(i-1))$

Step 3. Compute $\theta(i)=\theta(i-1)+\varepsilon\Theta(i-\frac{\varepsilon}{2})$

Step 4. Compute $\Theta(i)=\Theta(i-\frac{1}{2})-\frac{\varepsilon}{2}\nabla U(\theta(i)))$.

Step 5. Set $i=i+1$ and return to **Step 2**. Escape the loop when $i=L$, the desired number of steps.

[1] For other sampling strategies, see [2]
[2] On average, it takes around 10 ~ 11 hours to generate 1 million data points using a personal laptop with 5th Generation Intel® Core™ i7 Processors.

Combined with the leapfrog method, we have the following steps for HMC algorithm[3]:

Step 1. Begin with $i = 1$, an arbitrary initial point $\theta^{(i)}$, and time step T.

Step 2. Generate canonical momentum $\Theta^{(i)}$ from a multivariate normal distribution with unit variance.

Step 3. Using the leapfrog method, start from $\{\theta^{(i)}, \Theta^{(i)}\}$ and obtain $\{\theta^*, \Theta^*\}$ after a finite time period T.

Step 4. Compute the acceptance ratio[3] $a = \min\{e^{H(\theta^{(i)},\Theta^{(i)})-H(\theta^{(*)},\Theta^{(*)})}, 1\}$

Step 5. Pick a random number b uniformly from range $0 < b < 1$. If $a > b$, set $\theta^{(i+1)} = \theta^{(*)}$; else let $\theta^{(i+1)} = \theta^{(i)}$.

Step 6. Set $i = i + 1$ and return to **Step 2**. Escape the loop when i reaches the target number of sample points.

As mentioned previously, in order to implement HMC on quantum systems, we will need a parameterization method of density matrix ρ. In the following chapter, we discuss two possible solutions to this problem, namely the Cholesky decomposition and spectral decomposition of ρ.

[3]For high dimension problem, it is best to maintain an acceptance rate around 65%, by adjusting T and L.

Chapter 2

Random Density Matrices

A density matrix ρ, or density operator, is a matrix that can be used to describe a quantum system. It is a positive semidefinite, Hermitian operator of unit trace, *i.e.*,

$$\mathrm{Tr}\{\rho\} = 1, \text{ and } \lambda_i \geq 0, \tag{2.1}$$

where $\{\lambda_i\}$ are the eigenvalues of ρ.
For a density matrix that lives in a d-dimensional Hilbert space, there are $(d^2 - 1)$ independent real parameters. In this chapter, we demonstrate two possible parameterization methods for the density matrix.

2.1 Cholesky Decomposition

Every Hermitian, positive semidefinite density matrix ρ has a Cholesky decomposition taking the form

$$\rho = A^\dagger A \tag{2.2}$$

where A is an upper-triangular matrix with real and non-negative diagonal entries. Hence

$$\mathrm{Tr}\{A^\dagger A\} = \sum_{1 \leq j \leq k \leq d} |A_{jk}|^2 = 1 \tag{2.3}$$

That is, the moduli of elements of matrix A are points lying on a sphere of dimension $\frac{1}{2}d(d+1) - 1 = \frac{1}{2}(d+2)(d-1)$. This sphere can be parameterized by a set of angle parameters $\theta_1, \theta_2, \ldots, \theta_{\frac{1}{2}(d+2)(d-1)}$ with Cartesian coordinates $C_1, C_2, \ldots, C_{\frac{1}{2}(d+2)(d-1)}, S_{\frac{1}{2}(d+2)(d-1)}$ defined by

$$\begin{aligned} &C_1 = \cos\theta_1,\ S_1 = \sin\theta_1, \\ &C_k = S_{k-1}\cos\theta_k,\ S_k = S_{k-1}\sin\theta_k,\ \text{for } k = 2, 3, \ldots, \frac{1}{2}(d+2)(d-1). \end{aligned} \tag{2.4}$$

The upper-triangular matrix A is filled in with these Cartesian coordinates, which consists of $\frac{1}{2}d(d+1)$ number of entries in total, and its off-diagonal terms are further supplemented by phase factors

$$E_k = e^{-\mathrm{i}\theta_k},\ \text{for } k = \frac{1}{2}d(d+1), \ldots, d^2 - 1. \tag{2.5}$$

There are different ways of assigning the Cartesian coordinates and phase factors to matrix A, and they are all equally valid. Written down explicitly in component form, one possible way is given by

$$A_{ij} = \begin{cases} 0 & \text{if } i > j \\ C_{\frac{1}{2}j(j+1)} & \text{if } i = j < d \\ S_{\frac{1}{2}(d+2)(d-1)} & \text{if } i = j = d \\ C_m E_{m+n} & \text{if } i < j \end{cases} \tag{2.6}$$

with

$$m = \frac{1}{2}j(j-1) + i, \quad n = \frac{1}{2}(d+2)(d-1) - (j-1). \tag{2.7}$$

The density matrix ρ is now parameterized in terms of the set of independent variables $\{\theta_1, \theta_2, \ldots, \theta_{d^2-1}\}$. The probability of the kth detector clicking, p_k, is given by

$$p_k = \mathrm{Tr}\{A^\dagger A\, \Pi_k\} \tag{2.8}$$

From the probability space characterised by these probabilities $\{p_1, p_2, \ldots, p_K\}$, to our parameterisation space defined by variables $\{\theta_1, \theta_2, \ldots, \theta_l\}$, we have the Jacobian matrix

$$\frac{\partial p}{\partial \theta} = 2\,\mathrm{Re}\Big\{\mathrm{Tr}\Big\{A^\dagger \frac{\partial A}{\partial \theta}\,\Pi\Big\}\Big\} \tag{2.9}$$

or, in component form,

$$\Big\{\frac{\partial p}{\partial \theta}\Big\}_{ij} \equiv \frac{\partial p_i}{\partial \theta_j} = 2\,\mathrm{Re}\Big\{\mathrm{Tr}\Big\{A^\dagger \frac{\partial A}{\partial \theta_j}\,\Pi_i\Big\}\Big\} \tag{2.10}$$

The prior or posterior density in p can be expressed in terms of θ,

$$w(p) \to w(\theta) \equiv \Big(w(p)\Big|\frac{\partial p}{\partial \theta}\Big|\Big)\Big|_{p\,\text{in terms of }\theta} \propto \Big|\frac{\partial p}{\partial \theta}\Big| \tag{2.11}$$

where $|\frac{\partial p}{\partial \theta}|$ is the Jacobian determinant.
In order to execute the HMC algorithm introduced in Chapter 1, we also need the

gradient of "potential energy", $u_s(\theta) = \frac{\partial U(\theta)}{\partial \theta_s}$, which is proportional to

$$\mathrm{Tr}\Big\{\Big\{\frac{\partial p}{\partial \theta}\Big\}^{-1} \cdot 2\,\mathrm{Re}\Big\{\mathrm{Tr}\Big\{\Big(\frac{\partial A^\dagger}{\partial \theta_s}\frac{\partial A}{\partial \theta} + A^\dagger \frac{\partial}{\partial \theta_s}\frac{\partial A}{\partial \theta}\Big)\Pi\Big\}\Big\} \tag{2.12}$$

where $\{\frac{\partial p}{\partial \theta}\}^{-1}$ refers to the matrix inverse of Jacobian matrix.

2.1.1 Example: Single qubit, $d = 2$

To parameterize ρ for a single qubit, we have

$$A = \begin{pmatrix} C_1 & C_2 E_3 \\ 0 & S_2 \end{pmatrix} = \begin{pmatrix} \cos\theta_1 & \sin\theta_1 \cos\theta_2\, e^{i\theta_3} \\ 0 & \sin\theta_1 \sin\theta_2 \end{pmatrix} \tag{2.13}$$

and

$$\rho = A^\dagger A = \begin{pmatrix} \cos^2\theta_1 & \frac{1}{2}\sin(2\theta_1)\sin\theta_2\, e^{i\theta_3} \\ \frac{1}{2}\sin(2\theta_1)\sin\theta_2\, e^{-i\theta_3} & \sin^2\theta_1 \end{pmatrix} \tag{2.14}$$

The probabilities of corresponding Pauli matrices are given by

$$\begin{aligned} x &= \langle \sigma_x \rangle = \sin(2\theta_1)\sin\theta_2\cos\theta_3 \\ y &= \langle \sigma_y \rangle = \sin(2\theta_1)\sin\theta_2\sin\theta_3 \\ z &= \langle \sigma_z \rangle = \cos(2\theta_1) \end{aligned} \tag{2.15}$$

The Jacobian matrix is

$$\frac{\partial p}{\partial \theta} = \begin{pmatrix} 2\cos(2\theta_1)\cos\theta_2\cos\theta_3 & -\cos\theta_3\sin(2\,\theta_1)\sin\theta_2 & -\cos\theta_2\sin(2\,\theta_1)\sin\theta_3 \\ 2\cos(2\theta_1)\cos\theta_2\sin\theta_3 & -\sin(2\,\theta_1)\sin\theta_2\sin\theta_3 & \cos\theta_2\cos\theta_3\sin(2\,\theta_1) \\ -2\sin(2\,\theta_1) & 0 & 0 \end{pmatrix} \tag{2.16}$$

Its determinant, $\left|\frac{\partial p}{\partial \theta}\right|$, is found to be

$$\left|\frac{\partial p}{\partial \theta}\right| = \sin^3(2\,\theta_1)\sin(2\,\theta_2) \tag{2.17}$$

Finally, to implement HMC algorithm, we have the gradients of the potential energy

$$\begin{aligned} u_1(\theta) &= 6\cot(2\,\theta_1) \\ u_2(\theta) &= 2\cot(2\,\theta_2) \\ u_3(\theta) &= 0 \end{aligned} \tag{2.18}$$

2.1.2 Example: Qubit pair, $d = 4$

Following the definition from Equations 2.4 and 2.5, we can decompose the density matrix ρ for a qubit pair with

$$A = \begin{pmatrix} C_1 & C_2E_{10} & C_4E_{11} & C_7E_{13} \\ 0 & C_3 & C_5E_{12} & C_8E_{14} \\ 0 & 0 & C_6 & C_9E_{15} \\ 0 & 0 & 0 & S_9 \end{pmatrix} \tag{2.19}$$

There are 15 independent variables, $\theta_1, \theta_2, \ldots, \theta_{15}$. To compute the Jacobian matrix and the potential gradient from Equations 2.9 and 2.12, we will need to take first order, as well as second order derivatives of matrix A with respect to all these θ variables. Unfortunately, unlike the trivial example of previous single qubit case, we are unable to obtain analytic expressions at this stage. Instead, we develop a numerical approach to perform HMC algorithm with MATLAB®.

2.2 Spectral Decomposition

Spectral decomposition, or sometimes eigen-decomposition, is the factorization of a square matrix in terms of its eigenvalues and eigenvectors. For our Hermitian density matrix ρ, we can write

$$\rho = UDU^\dagger \tag{2.20}$$

where U is a unitary matrix whose columns are the eigenvectors of ρ, and D is a diagonal matrix of unit trace, formed by the eigenvalues of ρ.
The diagonal elements of matrix D are defined recursively with $(d-1)$ angle parameters $\{\alpha_1, \alpha_2, \ldots, \alpha_{d-1}\}$ as

$$\begin{aligned} &C_1 = \cos^2\alpha_1, \;\; S_1 = \sin^2\alpha_1, \\ &C_k = S_{k-1}\cos^2\alpha_k, \;\; S_k = S_{k-1}\sin^2\alpha_k, \;\; \text{for } k = 2, 3, \ldots, d-1. \end{aligned} \tag{2.21}$$

and matrix $D = \text{diag}\{C_1, C_2, \ldots, C_{d-1}, S_{d-1}\}$
The unitary matrix U is decomposed into a set of elementary unitary transformations in two-dimensional subspaces, with two independent variables $\{\theta, \varphi\}$ for each subspace. Every 2-d elementary unitary transformation is denoted as

$U^{(i,j)} = U^{(i,j)}(\theta_{ij}, \varphi_{ij})$, and constructs the following $(d-1)$ rotations:

$$\begin{aligned}
U_1 &\equiv U^{(1,2)}(\theta_{12}, \varphi_{12})\, U^{(1,3)}(\theta_{13}, \varphi_{13}) \cdots U^{(1,d)}(\theta_{1d}, \varphi_{1d}) \\
U_2 &\equiv U^{(2,3)}(\theta_{23}, \varphi_{23})\, U^{(2,4)}(\theta_{24}, \varphi_{24}) \cdots U^{(2,d)}(\theta_{2d}, \varphi_{2d}) \\
U_3 &\equiv U^{(3,4)}(\theta_{34}, \varphi_{34})\, U^{(3,5)}(\theta_{35}, \varphi_{35}) \cdots U^{(3,d)}(\theta_{3d}, \varphi_{3d}) \\
&\cdots \\
U_{d-1} &\equiv U^{((d-1),d)}(\theta_{(d-1)d}, \varphi_{(d-1)d})
\end{aligned} \tag{2.22}$$

The elementary unitary transformation $U^{(i,j)}(\theta, \varphi)$ has the following non-zero entries,

$$\begin{aligned}
U^{(i,j)}_{mm} &= 1, \quad \text{for } m = 1, 2, \ldots, d, \ \& \ m \neq i, j; \\
U^{(i,j)}_{ii} &= U^{(i,j)}_{jj} = \cos\theta; \\
U^{(i,j)}_{ij} &= e^{i\varphi} \sin\theta; \\
U^{(i,j)}_{ji} &= -e^{-i\varphi} \sin\theta.
\end{aligned} \tag{2.23}$$

The unitary matrix U is given by the product of $(d-1)$ rotations,

$$U = U_1 U_2 U_3 \cdots U_{(d-1)} \tag{2.24}$$

We now have $\frac{1}{2}d(d-1)$ number of θ and same amount of φ, contributing a total number of $d(d-1)$ independent variables for U. Together with the $(d-1)$ parameters in matrix D, we fulfill the demand of (d^2-1) independent parameters for our density matrix ρ.
For the θ, φ variables, the elements of Jacobian matrix are given by

$$\left\{\frac{\partial p}{\partial \phi}\right\}_{ij} \equiv \frac{\partial p_i}{\partial \phi_j} = 2\,\mathrm{Re}\Big\{\mathrm{Tr}\Big\{\frac{\partial U}{\partial \phi_j} D U^\dagger\, \Pi_i\Big\}\Big\} \tag{2.25}$$

where $\phi = \{\theta_1, \theta_2, \ldots, \theta_{\frac{1}{2}d(d-1)}, \varphi_1, \varphi_2, \ldots, \varphi_{\frac{1}{2}d(d-1)}\}$.
On the other hand, Jacobian elements for α variables are

$$\left\{\frac{\partial p}{\partial \alpha}\right\}_{ij} \equiv \frac{\partial p_i}{\partial \alpha_j} = \mathrm{Tr}\Big\{U \frac{\partial D}{\partial \alpha_j} U^\dagger \Pi_i\Big\} \tag{2.26}$$

The Jacobian matrix may seem to be a bit more complicated than that in Cholesky decomposition, as three types of variables are to be dealt with instead of two. However, it is much easier to realise in actual practice, due to the fact that matrix U is the product of a series of matrices with independent variables for their own. For instance, to compute $\frac{\partial U}{\partial \phi_k}$, we only need to replace matrix U_k with its derivative

$\frac{\partial U_k}{\partial \phi_k}$, leaving all the rest in the product unchanged, *i.e.*,

$$\frac{\partial U}{\partial \phi_k} = U_1 U_2 \cdots U_{k-1} \frac{\partial U_k}{\partial \phi_k} U_{k+1} \cdots U_{d-1} \tag{2.27}$$

This process is further simplified as a consequence of our way of defining the elementary unitary transformation. Taking a closer look of $U^{(i,j)}(\theta, \varphi)$ from Equations 2.23, we have

$$U^{(i,j)}(\theta,\varphi) = \begin{array}{c} \\ i\text{-th} \\ \\ j\text{-th} \\ \\ \end{array} \begin{array}{c} \begin{array}{ccccc} & i\text{-th} & & j\text{-th} & \end{array} \\ \begin{pmatrix} \cdots & \vdots & \cdots & \vdots & \cdots \\ \cdots & \cos\theta & \cdots & e^{i\varphi}\sin\theta & \cdots \\ \cdots & \vdots & \cdots & \vdots & \cdots \\ \cdots & -e^{-i\varphi}\sin\theta & \cdots & \cos\theta & \cdots \\ \cdots & \vdots & \cdots & \vdots & \cdots \end{pmatrix} \end{array} \tag{2.28}$$

That is, only four terms are involved when taking derivatives with respect to θ and φ, while the remaining entries are simply 0 or 1. Similarly, this ease of implementation extends to the second order derivatives as well.

2.2.1 Example: Single qubit, $d = 2$

Similar to Example 2.1.1, the single qubit spectral decomposition is the most fundamental and trivial. Density matrix ρ is decomposed into

$$\rho = U_1 D U_1^\dagger \tag{2.29}$$

where

$$U_1 = U^{(1,2)} = \begin{pmatrix} \cos\theta & e^{i\varphi}\sin\theta \\ -e^{-i\varphi}\sin\theta & \cos\theta \end{pmatrix}, \quad D = \begin{pmatrix} \cos^2\alpha & 0 \\ 0 & \sin^2\alpha \end{pmatrix} \tag{2.30}$$

Using Pauli matrices as POMs, the Jacobian determinant is found to be

$$\csc(2\alpha)\sin^2(4\alpha)\sin(2\theta) \tag{2.31}$$

The gradient of potential energy is therefore given by

$$\begin{aligned} u(\alpha) &= -2\cot(2\alpha) + 8\cot(4\alpha) \\ u(\theta) &= 2\cot(2\theta) \\ u(\varphi) &= 0 \end{aligned} \tag{2.32}$$

The HMC algorithm can now be executed accordingly.

Chapter 3

Applications

3.1 Preliminary

3.1.1 POM

Pauli matrices are used as POMs in previous examples for illustration purposes. When generating the samples studied in this chapter, we use tetrahedron measurements[4] instead, which are symmetric and informationally complete (SIC). For a single qubit, they are given by

$$\begin{aligned}
\Pi_1 &= \frac{1}{4}\left(1 + \sqrt{\frac{1}{3}}\sigma_x + \sqrt{\frac{2}{3}}\sigma_y\right) \\
\Pi_2 &= \frac{1}{4}\left(1 + \sqrt{\frac{1}{3}}\sigma_x - \sqrt{\frac{2}{3}}\sigma_y\right) \\
\Pi_3 &= \frac{1}{4}\left(1 - \sqrt{\frac{1}{3}}\sigma_x - \sqrt{\frac{2}{3}}\sigma_z\right) \\
\Pi_4 &= \frac{1}{4}\left(1 - \sqrt{\frac{1}{3}}\sigma_x + \sqrt{\frac{2}{3}}\sigma_z\right)
\end{aligned} \tag{3.1}$$

For multiple qubits, their tensor products are used as POMs.

3.1.2 Prior Density

Other than the flexibility of choosing different POMs, we can also perform sampling subject to different prior densities. In the following examples, we use three types of prior density $w_0(p)$. In *primitive prior*, the density is uniformly distributed

in p, *i.e.*,

$$w_{\text{primitive}}(p) = 1 \tag{3.2}$$

Another choice is the *Jeffreys prior*, which is

$$w_{\text{Jeffreys}}(p) = \frac{1}{\sqrt{p_1 p_2 \cdots p_K}} \tag{3.3}$$

Finally we have the *hedged prior*, given by

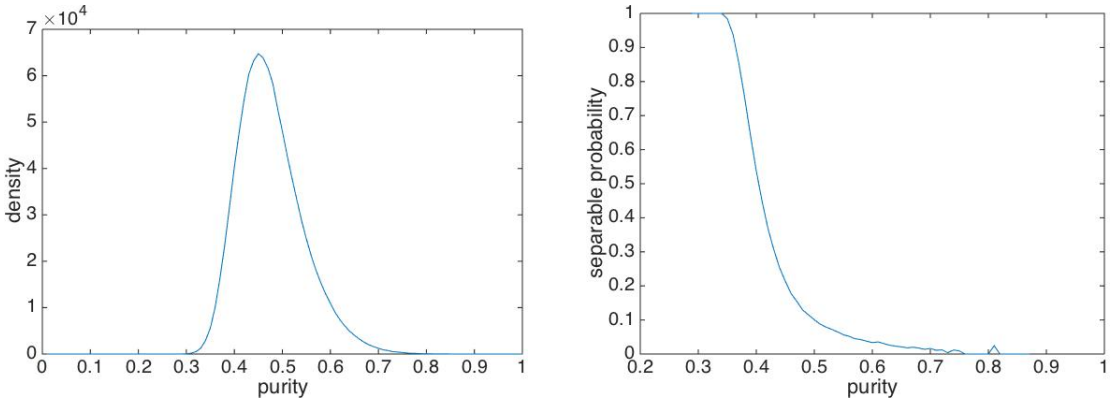
$$w_{\text{hedged}}(p) = \sqrt{p_1 p_2 \cdots p_K} \tag{3.4}$$

3.2 Purity

For a quantum state ρ, its purity $\xi(\rho)$ is a scalar quantity given by

$$\xi(\rho) \equiv \text{Tr}(\rho^2) \tag{3.5}$$

If we randomly pick up a 2-qubit state, what is the probability for it to have a certain purity value? And, for a 2-qubit state with certain purity value, what are the chances that it is separable?
To answer such questions, we first use HMC to randomly choose 1 million two-qubit states, using Cholesky decomposition and with respect to primitive prior. For each one of these sample states, its purity and separability[5] are then evaluated. The final results are shown below. It is seen clearly in Figure 3.1B that if $\xi(\rho) < 1/3$, then ρ is separable[6]. Here we successfully reproduce the same results as in [2], with a larger sample size.



(A) (B)

FIGURE 3.1: Sample of 1 million quantum states generated by Cholesky decomposition with respect to primitive prior density. (A) Prior density of quantum states with respect to purity. (B) Probability of separation as a function of purity.

3.3 Fidelity and Distance

Given two density matrices ρ_1 and ρ_2, their fidelity F is given by

$$F(\rho_1, \rho_2) = \mathrm{Tr}\left\{\sqrt{\sqrt{\rho_1}\rho_2\sqrt{\rho_1}}\right\} \tag{3.6}$$

On the other hand, we define their trace distance D as

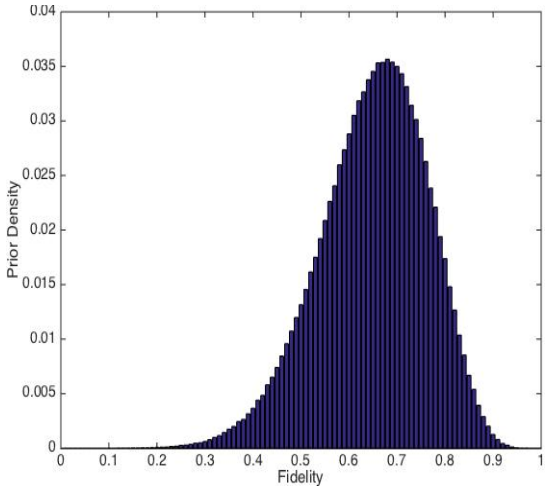
$$D(\rho_1, \rho_2) = \frac{1}{2}\,\mathrm{Tr}\left\{\sqrt{(\rho_1 - \rho_2)^\dagger(\rho_1 - \rho_2)}\right\} \tag{3.7}$$

And they satisfy the following inequalities[7],

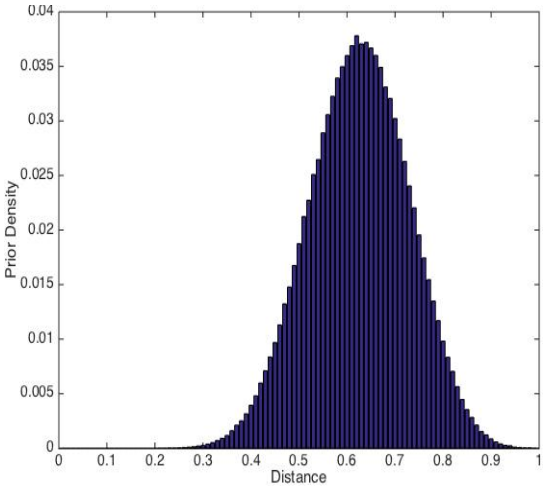
$$1 - F(\rho_1, \rho_2) \leq D(\rho_1, \rho_2) \leq \sqrt{1 - F(\rho_1, \rho_2)^2} \tag{3.8}$$

In this section, we first generate sample data with Cholesky decomposition method, based on certain prior density as introduced in Section 3.1.2. Two density matrices are then selected uniformly out of the sample, and their fidelity F as well as trace distance D are computed thereafter. MATLAB® code is attached in Appendix A.4.

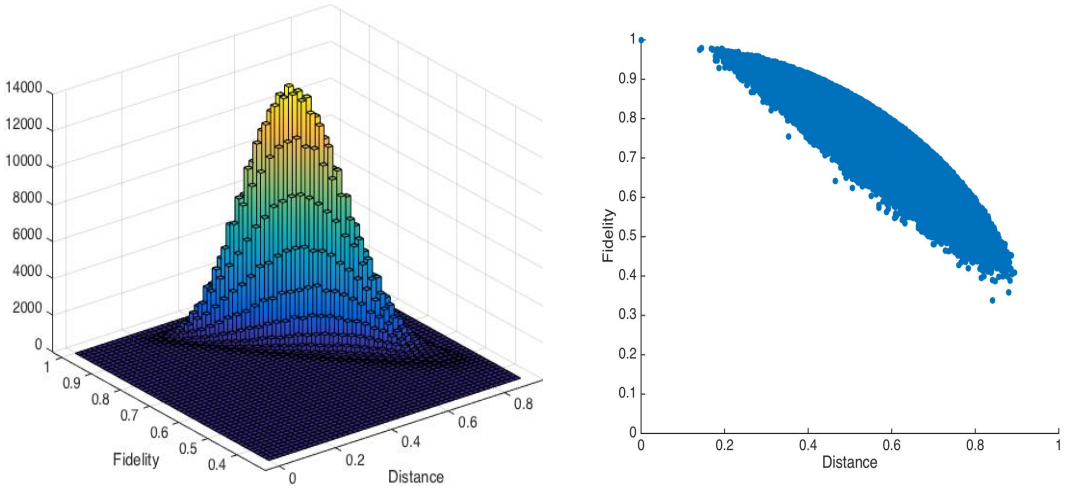
For primitive prior, we have the following results.



(A)



(B)



(C) (D)

FIGURE 3.2: 1 million sample two-qubit states with Cholesky decomposition and primitive prior density. Figure (A) and Figure (B) show the probability distribution of their fidelity and distance respectively. A 3-d histogram is shown in Figure (C). Figure (D) plots the fidelity and distance values for each sample state. The shape in (D) is nicely bounded by a unit radius circle.

It is clearly seen in Figure 3.2D that all data points lie within a circle with unit radius, and Fidelity F is above the straight line $(-D + 1)$. Both observations consist with the inequalities of Equation 3.8.

Similar observations can be made with Jeffreys prior (Figure 3.3D) and hedged prior (Figure 3.4D) as well, since the inequalities 3.8 are general results regardless of the choice of prior density.

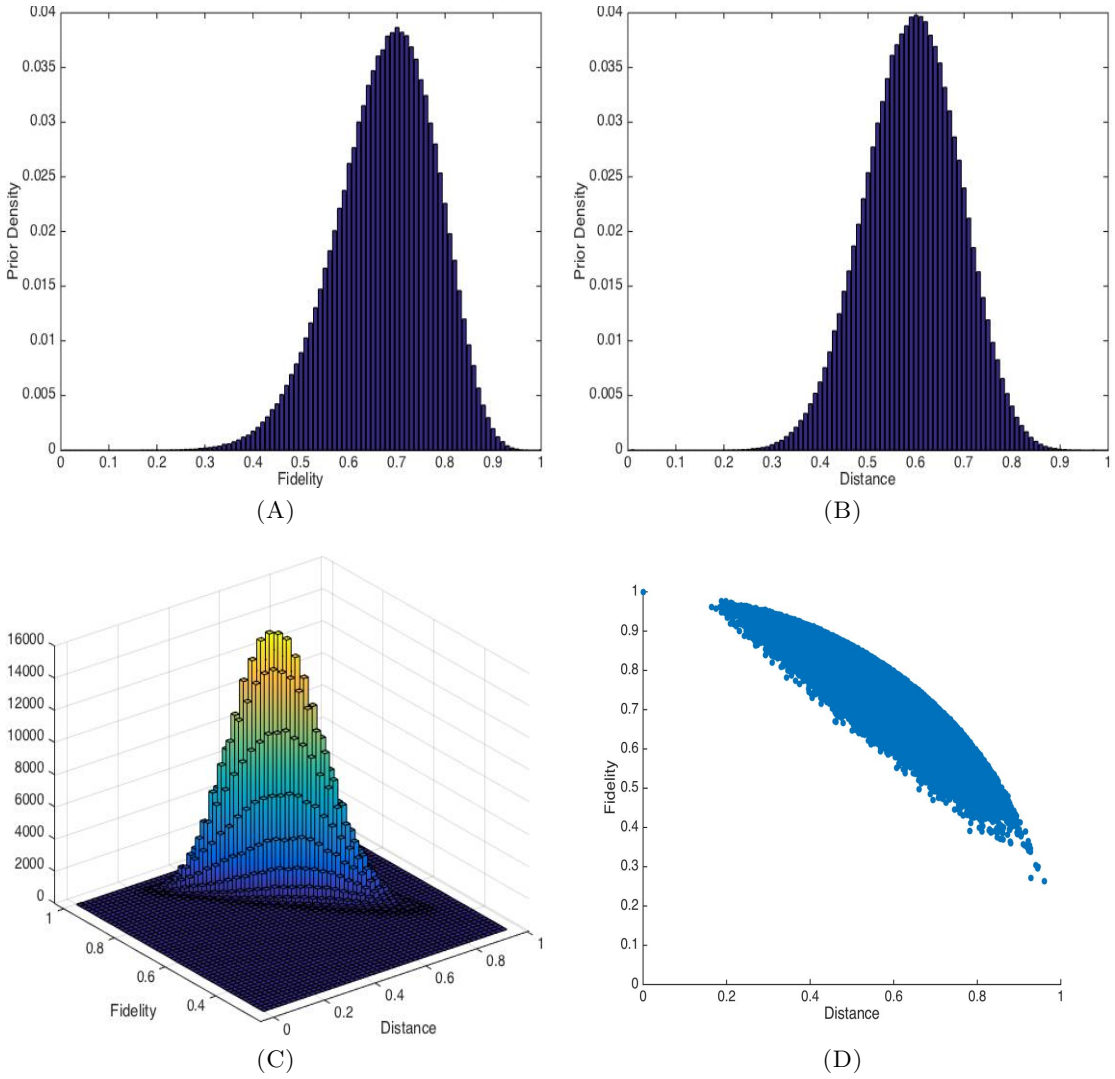


FIGURE 3.3: 1 million sample two-qubit states with Cholesky decomposition and Jeffreys prior. (A) and (B) show the probability distribution of fidelity and distance respectively. (C) is the 3-d histogram and (D) is the scatter plot for fidelity and distance values of each sample state.

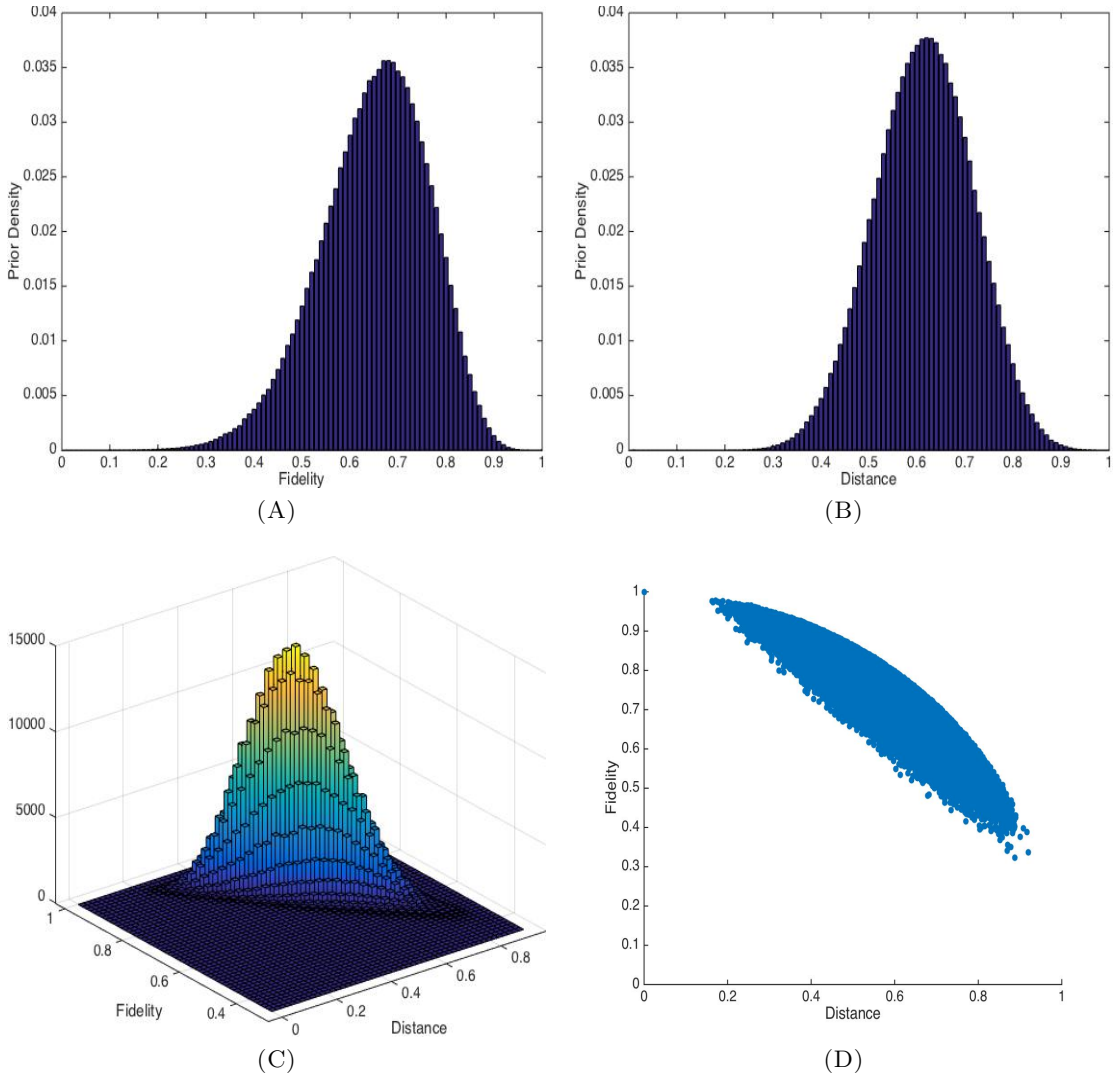


FIGURE 3.4: 1 million sample two-qubit states with Cholesky decomposition and hedged prior. (A) and (B) show the probability distribution of fidelity and distance respectively. (C) is the 3-d histogram and (D) is the scatter plot for fidelity and distance values of each sample state.

Chapter 4

Conclusion and Outlook

The main outcomes of this project are the MATLAB® code listed in Appendix A, which computes numerically the density matrix ρ, Jacobian determinant $|\frac{\partial p}{\partial \theta}|$, and gradients $u_s(\theta)$, for a given set of POM and angle variables used in Cholesky or spectral decomposition, with respect to any target prior density. Together with the HMC algorithm developed in [3], the code performs efficiently enough to generate 1 million random two-qubit states within around 10 ~ 11 hours.

In Chapter 1, we first introduced the concepts of size $S_{\mathcal{R}}$ and credibility $c_{\mathcal{R}}$ as in Equations 1.7 and 1.11, which served as motivations of developing HMC algorithm in order to estimate numerically the (d^2-1)-dimensional integral. Then we directly wrote out the HMC algorithm as shown in Section 1.3.2. Its formal and detailed introduction can be found in [3].

Two possible decomposition methods for the density matrix ρ, namely the Cholesky Decomposition and Spectral Decomposition, were discussed in Chapter 2. In Cholesky Decomposition, density matrix ρ is decomposed into the product of an upper-triangular matrix A with its Hermitian conjugate, *i.e.*, $\rho = A^\dagger A$, and each element of matrix A is a function of $(d^2 - 1)$ independent angle variables $\{\theta_1, \theta_2, \ldots, \theta_{d^2-1}\}$. The main challenge was to compute the first order and second order derivatives of A with respect to each one of these variables, as required in Equations 2.9 and 2.12. This part of MATLAB® code is attached in Appendix A.1 and A.2.

In contrast to Cholesky Decomposition, where one matrix A consists of all the independent variables, making evaluating its derivatives rather difficult, Spectral Decomposition has the nice properties of defining and assigning each individual variable into its own matrix. By replacing the elementary matrix with its derivative, the overall derivative and thus the Jocabian matrix can be easily computed,

as illustrated in Equation 2.27. An example of performing Spectral Decomposition with primitive prior is listed in Appendix A.3.

In Chapter 3, we studied some properties of the sample generated by the program. The probability distribution of purity, fidelity or distance of two-qubit states could be obtained with ease. It is also possible to analyze some other quantities such as the separation probability, relations between fidelity and distance, *etc.*

Although our MATLAB® program was written in such a manner as to compute numerically for any dimension, we were sampling in the 15-dimensional two-qubit state space throughout this thesis. For higher dimensional situations, qualitatively speaking, there will be too many *cosine* and *sine* functions of independent angle variables multiplied together, which may result in numbers that are too small for MATLAB® to handle properly. Hence further improvements need to be made for 63-dimensional three-qubit state space as well as even higher dimensional problems.

Appendix A

MATLAB code for two-qubit states

A.1 Cholesky Decomposition with Primitive Prior

```
% input angle variables q, matrix dimension d, and POM Q
% output matrix A used in Cholesky Decomposition, Jacobian determinant JacDet, and potential gradients u
% no need to output probabilities prob, which are all 1 in primitive prior

function [A, JacDet, u] = cholesky_2qb_flat(q,d,Q)

% d=4;
nt=d*(d+1)/2-1; % theta=9
nf=d*(d-1)/2; % phi=6
num=d^2-1;
% num=nt+nf=d^2-1=15

% indices that will be used repeatedly
ind = [d*nt,d*nf,nt+nf,d*(nt+nf),nt*nf,d*nt*nf,nt*(1+nt)/2,d*nt*(1+nt)/2];

% index matrices: indM1, indM2, and so on.
indM1 = tril(ones(d));
indM2 = tril(ones(d),-1);

% indM3, indM4 and indM5 are used to compute partial traces
indM3 = reshape(reshape(bsxfun(@plus,(0:d^2-2)*(d^4-d^2),reshape(bsxfun(@plus,(0:d-1)*(d^3-d+1),(1:d:d^3-d*2+1).'),1,d^3-d).'),d^2-1,d^3-d).',d,(d^2-1)^2);
indM4 = reshape(reshape(bsxfun(@plus,(0:14)*3600,reshape(bsxfun(@plus,(0:d-1)*901,(1:d:897).'),1,900).'),225,60).',4,3375);
indM5 = bsxfun(@plus,(0:14)*225,bsxfun(@plus,(0:14)*16,1.').');

t=q(1:nt,1)';
f=q(nt+1:num,1)';

sint = sin(t); cost = cos(t); tant = tan(t); cott = cot(t);
```

```
29 expfp = exp(1i*f); expfm = exp(-1i*f);
30
31 % Create x, which are |Ajk| and lie on a sphere
32 x_temp = [1,cumprod(sint)].*[cost,1];
33 temp = ones(d);
34 temp(indM2 == 1) = temp(indM2 == 1)'.*expfm;
35 x = x_temp'.*nonzeros(tril(temp));
36
37 % cA is complex conjugate transpose of matrix A
38 % Create cA using complex conjugate of x
39 cA = zeros(d);
40 cA(indM1 == 1) = conj(x);
41 % Create matrix A
42 % density matrix rho = A'*A = cA*A
43 A = cA';
44
45 % derivative of x wrt theta
46 dxdt_temp = (x(2:nt+1)*cott).';
47 dxdt_temp(tril(true(nt),-1)==1) = 0;
48 dxdt = [zeros(nt,1),dxdt_temp];
49 temp = ones(d);
50 temp(indM2 == 1) = temp(indM2 == 1)'.*expfm;
51 temp = temp(indM1==1);
52 dxdt(logical(eye(nt))) = -cumprod(sint)'.*temp(1:nt);
53
54 % derivative of matrix A wrt theta
55 % Create dAdt using transpose of dxdt
56 dAdt = zeros(d,ind(1));
57 dAdt(repmat(indM1,1,nt) == 1) = dxdt.';
58 dAdt = dAdt.';
59
60 % here dxdf_temp is not yet derivative of x wrt phi
61 dxdf_temp = zeros(nt+1);
62 dxdf_temp(eye(nt+1)==1) = -1i*indM2(tril(true(d)));
63 dxdf_temp(~any(dxdf_temp,2),:) = [];
64
65 % 2nd order derivative of x wrt the same phi
66 ddxdff = -1i*dxdf_temp;
67 ddxdff = bsxfun(@times,ddxdff,x.');
68
69 % 2nd order derivative of A wrt the same phi
70 ddAdff = zeros(d,ind(2));
71 ddAdff(repmat(indM1,1,nf) == 1) = ddxdff.';
72 ddAdff = ddAdff.';
73
74 % y is 2nd order derivative of x wrt theta and phi
75 y = zeros(ind(5),10);
76 for i = 1:9
77     y(6*i-5:1:6*i,:) = y(6*i-5:1:6*i,:) + bsxfun(@times,dxdf_temp,dxdt(i,:));
78 end
79
80 % derivative of A wrt theta and phi
81 ddAdfdt = zeros(d,ind(6));
82 ddAdfdt(repmat(indM1,1,ind(5)) == 1) = y.';
83 ddAdfdt = ddAdfdt.';
```

```
84
85  % derivative of x wrt phi
86  dxdf = bsxfun(@times,dxdf_temp,x.');
87
88  % derivative of matrix A wrt phi
89  % Create dAdt using transpose of dxdf
90  dAdf = zeros(d,ind(2));
91  dAdf(repmat(indM1,1,nf) == 1) = dxdf.';
92  dAdf = dAdf.';
93
94  % Put dAdt and dAdf together to form dAdm
95  dAdm = [dAdt;dAdf];
96  dAdm = reshape(permute(reshape(dAdm.',d,d,ind(3)),[2,1,3]),d,ind(4));
97
98  % Pre-multiply dAdm with cA and get cAdAdm
99  cAdAdm = cA*dAdm;
100 cAdAdm = reshape(permute(reshape(cAdAdm,d,d,ind(3)),[1,3,2]),ind(4),d);
101
102 % Multiply cAdAdm with POMs
103 dpdm_temp = cAdAdm*Q(:,1:60);
104
105 % Take partial traces and their real part, then times 2 to get Jacobian dpdm
106 dpdm = reshape(2*real(sum(dpdm_temp(indM3))),d^2-1,d^2-1);
107
108 % determinant of Jacobian matrix dpdm
109 JacDet = det(dpdm);
110
111 % z is 2nd order derivative of x wrt theta
112 z = zeros(ind(7),10);
113 for i = 1:8
114     temp1 = cott(i)*triu(ones(9-i,10),i+1);
115     temp1 = times(temp1,repmat(cott(i+1:9).',1,10));
116
117     temp2 = zeros(9-i,10);
118     temp2(i*(9-i)+1:10-i:numel(temp2)) = ones(1,9-i); % = cott(2)*tant(3:9)
119     temp2 = times(temp2,repmat(-tant(i+1:9).',1,10));
120     temp2 = cott(i)*temp2;
121
122     z(-i^2/2+21*i/2-8:1:-i^2/2+19*i/2,:) = z(-i^2/2+21*i/2-8:1:-i^2/2+19*i/2,:)+
    temp1+temp2;
123 end
124 z = repmat(x.',ind(7),1).*z;
125
126 z2 = -1*triu(ones(9,10));
127 z2 = repmat(x.',9,1).*z2;
128
129 z(~any(z,2),:) = z2;
130
131 % 2nd order derivative of matrix A wrt theta
132 % Create dAdtdt using transpose of z
133 ddAdtdt = zeros(d,ind(8));
134 ddAdtdt(repmat(indM1,1,ind(7)) == 1) = z.';
135 ddAdtdt = ddAdtdt.';
136
137 % % Put ddAdtdt, ddAdfdt and ddAdff together to form ddAdmdm
```

```
% ddAdmdm = [ddAdtdt;ddAdfdt;ddAdff];
% ddAdmdm = reshape(permute(reshape(ddAdmdm.',d,d,105),[2,1,3]),d,d*105);
% % Pre-multiply ddAdmdm with cA and get cddAdmdm
% cddAdmdm = cA*ddAdmdm;

% cA*ddAdtdt
cddAdtdt = cA*reshape(permute(reshape(ddAdtdt.',d,d,ind(7)),[2,1,3]),d,ind(8));
a1Cell = mat2cell(cddAdtdt,d,d*(9:-1:1));
A1 = zeros(36);
for i = 1:9
    A1(d*i-3:1:d*i,:) = A1(d*i-3:1:d*i,:) + [repmat(zeros(d),1,i-1),a1Cell{i}];
end

cddAdtdt_permute = reshape(permute(reshape(cddAdtdt,d,d,ind(7)),[1,3,2]),ind(8),d
    );
a2Cell = mat2cell(cddAdtdt_permute,d*(9:-1:1),d);
A2 = zeros(36);
for i = 1:9
    A2(:,d*i-3:1:d*i) = A2(:,d*i-3:1:d*i) + [repmat(zeros(d),i-1,1);a2Cell{i}];
end

a3Cell = cell(1,nt);
for i = 1:9
   a3Cell{i} = a2Cell{i}(1:d,1:d);
end
A3 = blkdiag(a3Cell{:});

sumA = A1+A2-A3;

% cA*ddAdfdt
cddAdfdt = cA*reshape(permute(reshape(ddAdfdt.',d,d,ind(5)),[2,1,3]),d,ind(6));
bCell = mat2cell(cddAdfdt,d,ind(2)*ones(1,nt));
cddAdfdt_permute = reshape(permute(reshape(cddAdfdt,d,d,ind(5)),[1,3,2]),ind(6),d
    );
cCell = mat2cell(cddAdfdt_permute,ind(2)*ones(1,nt),d);
B = zeros(ind(1),ind(2));C = zeros(ind(2),ind(1));
for i = 1:9
   B(d*i-d+1:1:4*i,:) = B(d*i-d+1:1:d*i,:) + bCell{i};
   C(:,d*i-d+1:1:4*i) = C(:,d*i-d+1:1:d*i) + cCell{i};
end

% cA*ddAdff
cddAdff = cA*reshape(permute(reshape(ddAdff.',d,d,nf),[2,1,3]),d,ind(2));
dCell = mat2cell(cddAdff,d,d*ones(1,nf));
D = blkdiag(dCell{:});

% combine sumA B C D in the following way and add to dAdm' * dAdm;
% | sumA  B |
% |   C   D |
mat_1 = [sumA,B;C,D] + dAdm'*dAdm;  % a 60-by-60 matrix
mat_2 = cell2mat(mat2cell(mat_1,ind(4),d*ones(1,ind(3))).');  % 900-by-4
mat_3 = mat_2*Q(:,1:60); % multiply mat_2 with POMs; 900-by-60

% Take partial traces of mat_3 and their real part, then times 2
mat_4 = reshape(2*real(sum(mat_3(indM4))),ind(3),ind(3)^2);
```

```
191
192 inv_dpdm = eye(d^2-1)/dpdm; % inv_dpdm = inv(dpdm);
193
194 mat_5 = inv_dpdm*mat_4;
195
196 % Partial traces of mat_5 yields the gradients
197 u = sum(mat_5(indM5));
198 u(nt+1:nt+nf) = zeros(1,nf);
```

A.2 Cholesky Decomposition with Jeffreys Prior or Hedged Prior

```
% input angle variables q, matrix dimension d, and POM Q
% output matrix A used in Cholesky Decomposition, probabilities prob, Jacobian determinant JacDet, and potential gradients u.

function [A, prob, JacDet, u] = cholesky_2qb_non_flat(q,d,Q)

% d=4;
nt=d*(d+1)/2-1; % theta=9
nf=d*(d-1)/2; % phi=6
num=d^2-1;
% num=nt+nf=d^2-1=15

% indices that will be used repeatedly
ind = [d*nt,d*nf,nt+nf,d*(nt+nf),nt*nf,d*nt*nf,nt*(1+nt)/2,d*nt*(1+nt)/2];

% index matrices: indM1, indM2, and so on.
indM1 = tril(ones(d));
indM2 = tril(ones(d),-1);

% indM3, indM4, indM5 and indM6 are used to compute partial traces
indM3 = reshape(reshape(bsxfun(@plus,(0:d^2-2)*(d^4-d^2),reshape(bsxfun(@plus,(0:d-1)*(d^3-d+1),(1:d:d^3-d*2+1).'),1,d^3-d).'),d^2-1,d^3-d).',d,(d^2-1)^2);
indM4 = reshape(reshape(bsxfun(@plus,(0:14)*3600,reshape(bsxfun(@plus,(0:d-1)*901,(1:d:897).'),1,900).'),225,60).',4,3375);
indM5 = bsxfun(@plus,(0:14)*225,bsxfun(@plus,(0:14)*16,1.').');
indM6 = bsxfun(@plus,(0:14)*16,bsxfun(@plus,(0:3)*5,1.').');

t=q(1:nt,1)';
f=q(nt+1:num,1)';

sint = sin(t); cost = cos(t); tant = tan(t); cott = cot(t);
expfp = exp(1i*f); expfm = exp(-1i*f);

% Create x, which are |Ajk| and lie on a sphere
x_temp = [1,cumprod(sint)].*[cost,1];
temp = ones(d);
temp(indM2 == 1) = temp(indM2 == 1)'.*expfm;
x = x_temp'.*nonzeros(tril(temp));

% cA is complex conjugate transpose of matrix A
% Create cA using complex conjugate of x
cA = zeros(d);
cA(indM1 == 1) = conj(x);
% Create matrix A
A = cA';
% density matrix rho
rho = cA*A;
% probabilities
prob = rho*Q(:,1:60);
prob = sum(prob(indM6));

```

```
% derivative of x wrt theta
dxdt_temp = (x(2:nt+1)*cott).';
dxdt_temp(tril(true(nt),-1)==1) = 0;
dxdt = [zeros(nt,1),dxdt_temp];
temp = ones(d);
temp(indM2 == 1) = temp(indM2 == 1).'.*expfm;
temp = temp(indM1==1);
dxdt(logical(eye(nt))) = -cumprod(sint).'.*temp(1:nt);

% derivative of matrix A wrt theta
% Create dAdt using transpose of dxdt
dAdt = zeros(d,ind(1));
dAdt(repmat(indM1,1,nt) == 1) = dxdt.';
dAdt = dAdt.';

% here dxdf_temp is not yet derivative of x wrt phi
dxdf_temp = zeros(nt+1);
dxdf_temp(eye(nt+1)==1) = -1i*indM2(tril(true(d)));
dxdf_temp(~any(dxdf_temp,2),:) = [];

% 2nd order derivative of x wrt the same phi
ddxdff = -1i*dxdf_temp;
ddxdff = bsxfun(@times,ddxdff,x.');

% 2nd order derivative of A wrt the same phi
ddAdff = zeros(d,ind(2));
ddAdff(repmat(indM1,1,nf) == 1) = ddxdff.';
ddAdff = ddAdff.';

% y is 2nd order derivative of x wrt theta and phi
y = zeros(ind(5),10);
for i = 1:9
    y(6*i-5:1:6*i,:) = y(6*i-5:1:6*i,:) + bsxfun(@times,dxdf_temp,dxdt(i,:));
end

% derivative of A wrt theta and phi
ddAdfdt = zeros(d,ind(6));
ddAdfdt(repmat(indM1,1,ind(5)) == 1) = y.';
ddAdfdt = ddAdfdt.';

% derivative of x wrt phi
dxdf = bsxfun(@times,dxdf_temp,x.');

% derivative of matrix A wrt phi
% Create dAdt using transpose of dxdf
dAdf = zeros(d,ind(2));
dAdf(repmat(indM1,1,nf) == 1) = dxdf.';
dAdf = dAdf.';

% Put dAdt and dAdf together to form dAdm
dAdm = [dAdt;dAdf];
dAdm = reshape(permute(reshape(dAdm.',d,d,ind(3)),[2,1,3]),d,ind(4));

% Pre-multiply dAdm with cA and get cAdAdm
cAdAdm = cA*dAdm;
```

```
104 cAdAdm = reshape(permute(reshape(cAdAdm,d,d,ind(3)),[1,3,2]),ind(4),d);
105
106 % Multiply cAdAdm with POMs
107 dpdm_temp = cAdAdm*Q(:,1:60);
108
109 % Take partial traces and their real part, then times 2 to get Jacobian dpdm
110 dpdm = reshape(2*real(sum(dpdm_temp(indM3))),d^2-1,d^2-1);
111
112 % determinant of Jacobian matrix dpdm
113 JacDet = det(dpdm);
114
115 % z is 2nd order derivative of x wrt theta
116 z = zeros(ind(7),10);
117 for i = 1:8
118     temp1 = cott(i)*triu(ones(9-i,10),i+1);
119     temp1 = times(temp1,repmat(cott(i+1:9).',1,10));
120
121     temp2 = zeros(9-i,10);
122     temp2(i*(9-i)+1:10-i:numel(temp2)) = ones(1,9-i); % = cott(2)*tant(3:9)
123     temp2 = times(temp2,repmat(-tant(i+1:9).',1,10));
124     temp2 = cott(i)*temp2;
125
126     z(-i^2/2+21*i/2-8:1:-i^2/2+19*i/2,:) = z(-i^2/2+21*i/2-8:1:-i^2/2+19*i/2,:)+
        temp1+temp2;
127 end
128 z = repmat(x.',ind(7),1).*z;
129
130 z2 = -1*triu(ones(9,10));
131 z2 = repmat(x.',9,1).*z2;
132
133 z(~any(z,2),:) = z2;
134
135 % 2nd order derivative of matrix A wrt theta
136 % Create dAdtdt using transpose of z
137 ddAdtdt = zeros(d,ind(8));
138 ddAdtdt(repmat(indM1,1,ind(7)) == 1) = z.';
139 ddAdtdt = ddAdtdt.';
140
141 % % Put ddAdtdt, ddAdfdt and ddAdff together to form ddAdmdm
142 % ddAdmdm = [ddAdtdt;ddAdfdt;ddAdff];
143 % ddAdmdm = reshape(permute(reshape(ddAdmdm.',d,d,105),[2,1,3]),d,d*105);
144 % % Pre-multiply ddAdmdm with cA and get cddAdmdm
145 % cddAdmdm = cA*ddAdmdm;
146
147 % cA*ddAdtdt
148 cddAdtdt = cA*reshape(permute(reshape(ddAdtdt.',d,d,ind(7)),[2,1,3]),d,ind(8));
149 a1Cell = mat2cell(cddAdtdt,d,d*(9:-1:1));
150 A1 = zeros(36);
151 for i = 1:9
152     A1(d*i-3:1:d*i,:) = A1(d*i-3:1:d*i,:) + [repmat(zeros(d),1,i-1),a1Cell{i}];
153 end
154
155 cddAdtdt_permute = reshape(permute(reshape(cddAdtdt,d,d,ind(7)),[1,3,2]),ind(8),d
    );
156 a2Cell = mat2cell(cddAdtdt_permute,d*(9:-1:1),d);
```

```
157 A2 = zeros(36);
158 for i = 1:9
159     A2(:,d*i-3:1:d*i) = A2(:,d*i-3:1:d*i) + [repmat(zeros(d),i-1,1);a2Cell{i}];
160 end
161
162 a3Cell = cell(1,nt);
163 for i = 1:9
164    a3Cell{i} = a2Cell{i}(1:d,1:d);
165 end
166 A3 = blkdiag(a3Cell{:});
167
168 sumA = A1+A2-A3;
169
170 % cA*ddAdfdt
171 cddAdfdt = cA*reshape(permute(reshape(ddAdfdt.',d,d,ind(5)),[2,1,3]),d,ind(6));
172 bCell = mat2cell(cddAdfdt,d,ind(2)*ones(1,nt));
173 cddAdfdt_permute = reshape(permute(reshape(cddAdfdt,d,d,ind(5)),[1,3,2]),ind(6),d
        );
174 cCell = mat2cell(cddAdfdt_permute,ind(2)*ones(1,nt),d);
175 B = zeros(ind(1),ind(2));C = zeros(ind(2),ind(1));
176 for i = 1:9
177    B(d*i-d+1:1:4*i,:) = B(d*i-d+1:1:d*i,:) + bCell{i};
178    C(:,d*i-d+1:1:4*i) = C(:,d*i-d+1:1:d*i) + cCell{i};
179 end
180
181 % cA*ddAdff
182 cddAdff = cA*reshape(permute(reshape(ddAdff.',d,d,nf),[2,1,3]),d,ind(2));
183 dCell = mat2cell(cddAdff,d,d*ones(1,nf));
184 D = blkdiag(dCell{:});
185
186 % combine sumA B C D in the following way and add to dAdm' * dAdm;
187 % | sumA  B |
188 % |   C   D |
189 mat_1 = [sumA,B;C,D] + dAdm'*dAdm;  % a 60-by-60 matrix
190 mat_2 = cell2mat(mat2cell(mat_1,ind(4),d*ones(1,ind(3))).');  % 900-by-4
191 mat_3 = mat_2*Q(:,1:60); % multiply mat_2 with POMs; 900-by-60
192
193 % Take partial traces of mat_3 and their real part, then times 2
194 mat_4 = reshape(2*real(sum(mat_3(indM4))),ind(3),ind(3)^2);
195
196 inv_dpdm = eye(d^2-1)/dpdm; % inv_dpdm = inv(dpdm);
197
198 mat_5 = inv_dpdm*mat_4;
199
200 % Partial traces of mat_5 yields the gradients
201 u = sum(mat_5(indM5));
202 u(nt+1:nt+nf) = zeros(1,nf);
```

A.3 Spectral Decomposition with Primitive Prior

```
% input angle variables q, matrix dimension d, and POM Q
% output matrix U, matrix D used in Spectral Decomposition, Jacobian determinant
    JacDet, and potential gradients u
% no need to output probabilities prob, which are all 1 in primitive prior

function [U,  D, JacDet, u] = spect_2qb_flat(q,d,Q)

% d=4;
nft = d*(d-1)/2; % number of theta and phi are the same
na = d-1;        % number of alpha
num = d^2-1;
% num=nft+nft+na=d^2-1=15

t = q(1:nft,1)';
f = q(nft+1:nft+nft,1)';
a = q(nft+nft+1:num,1)';

sint = sin(t); cost = cos(t);
tant = tan(t); cott = cot(t);
expfp = exp(1i*f);  expfm = exp(-1i*f);
sina = sin(a); cosa = cos(a);
tana = tan(a); cota = cot(a);

% j,k indices for theta, phi and matrix E
ind = zeros(nft,2);
temp = 1;
for j = 1:d-1
    for k = (j+1):d
        ind(temp,:)=[j,k];
        temp=temp+1;
    end
end

E = zeros(d,d,nft);       % E_i is a matrix of theta_i and phi_i
dEdt = zeros(d,d,nft);    % 1st order derivative of E_i wrt theta_i
dEdf = zeros(d,d,nft);    % 1st order derivative of E_i wrt phi_i
d2Edtt = zeros(d,d,nft);  % 2nd order derivative of E_i wrt theta_i
d2Edft = zeros(d,d,nft);  % 2nd order derivative of E_i wrt theta_i and phi_i
d2Edff = zeros(d,d,nft);  % 2nd order derivative of E_i wrt phi_i
for i = 1:nft
    % E_i is a matrix of theta_i and phi_i
    E(:,:,i) = eye(d);
    E(ind(i,1),ind(i,1),i) = cost(i);
    E(ind(i,2),ind(i,2),i) = cost(i);
    E(ind(i,1),ind(i,2),i) = expfp(i)*sint(i);
    E(ind(i,2),ind(i,1),i) = -expfm(i)*sint(i);

    % 1st order derivative of E_i wrt theta_i
    dEdt(ind(i,1),ind(i,1),i) = -sint(i);
    dEdt(ind(i,2),ind(i,2),i) = -sint(i);
    dEdt(ind(i,1),ind(i,2),i) = expfp(i)*cost(i);
    dEdt(ind(i,2),ind(i,1),i) = -expfm(i)*cost(i);

```

```
53      % 1st order derivative of E_i wrt phi_i
54      dEdf(ind(i,1),ind(i,2),i) = 1i*E(ind(i,1),ind(i,2),i);
55      dEdf(ind(i,2),ind(i,1),i) = -1i*E(ind(i,2),ind(i,1),i);
56
57      % 2nd order derivative of E_i wrt theta_i
58      d2Edtt(ind(i,1),ind(i,1),i) = -E(ind(i,1),ind(i,1),i);
59      d2Edtt(ind(i,2),ind(i,2),i) = -E(ind(i,2),ind(i,2),i);
60      d2Edtt(ind(i,1),ind(i,2),i) = -E(ind(i,1),ind(i,2),i);
61      d2Edtt(ind(i,2),ind(i,1),i) = - E(ind(i,2),ind(i,1),i);
62
63      % 2nd order derivative of E_i wrt theta_i and phi_i
64      d2Edft(ind(i,1),ind(i,2),i) = 1i*dEdt(ind(i,1),ind(i,2),i);
65      d2Edft(ind(i,2),ind(i,1),i) = -1i*dEdt(ind(i,2),ind(i,1),i);
66
67      % 2nd order derivative of E_i wrt phi_i
68      d2Edff(ind(i,1),ind(i,2),i) = -E(ind(i,1),ind(i,2),i);
69      d2Edff(ind(i,2),ind(i,1),i) = -E(ind(i,2),ind(i,1),i);
70  end
71
72  % Product of matrices E
73  % U = E_1 * E_2 * E_3 * E_4 * ... * E_nft
74  U = eye(d);
75  for i = 1:nft
76      U = U*E(:,:,i);
77  end
78  cU = ctranspose(U);
79
80  dUdm = zeros(d,d,nft+nft);  % 1st order derivative of U wrt theta and phi
81  cdUdm = zeros(d,d,nft+nft);
82  for i = 1:nft
83      dUdm(:,:,i) = eye(d);
84      for j = 1:i-1
85          dUdm(:,:,i) = dUdm(:,:,i)*E(:,:,j);
86      end
87      dUdm(:,:,i+nft) = dUdm(:,:,i)*dEdf(:,:,i);  % wrt phi
88      dUdm(:,:,i) = dUdm(:,:,i)*dEdt(:,:,i);  % wrt theta
89      for j = i+1:nft
90          dUdm(:,:,i) = dUdm(:,:,i)*E(:,:,j); % wrt theta
91          dUdm(:,:,i+nft) = dUdm(:,:,i+nft)*E(:,:,j); % wrt phi
92      end
93      cdUdm(:,:,i) = ctranspose(dUdm(:,:,i));
94      cdUdm(:,:,i+nft) = ctranspose(dUdm(:,:,i+nft));
95  end
96
97  d2Udmm = zeros(d,d,nft+nft,nft+nft);  % 2nd order derivative wrt theta and phi
98  for i = 1:nft
99      d2Udmm(:,:,i,i) = eye(d);  % 2nd order derivative of U wrt same theta
100     for j = 1:i-1
101         d2Udmm(:,:,i,i) = d2Udmm(:,:,i,i)*E(:,:,j);
102         d2Udmm(:,:,j,i) = eye(d);
103         d2Udmm(:,:,j,i+nft) = eye(d);
104         d2Udmm(:,:,j+nft,i+nft) = eye(d);
105         for k = 1:j-1
106             d2Udmm(:,:,j,i) = d2Udmm(:,:,j,i)*E(:,:,k);
107             d2Udmm(:,:,j,i+nft) = d2Udmm(:,:,j,i+nft)*E(:,:,k);
```

```
108             d2Udmm(:,:,j+nft,i+nft) = d2Udmm(:,:,j+nft,i+nft)*E(:,:,k);
109         end
110         d2Udmm(:,:,j,i) = d2Udmm(:,:,j,i)*dEdt(:,:,j);
111         d2Udmm(:,:,j,i+nft) = d2Udmm(:,:,j,i+nft)*dEdt(:,:,j);
112         d2Udmm(:,:,j+nft,i+nft) = d2Udmm(:,:,j+nft,i+nft)*dEdf(:,:,j);
113         for k = j+1:i-1
114             d2Udmm(:,:,j,i) = d2Udmm(:,:,j,i)*E(:,:,k);
115             d2Udmm(:,:,j,i+nft) = d2Udmm(:,:,j,i+nft)*E(:,:,k);
116             d2Udmm(:,:,j+nft,i+nft) = d2Udmm(:,:,j+nft,i+nft)*E(:,:,k);
117         end
118         d2Udmm(:,:,j,i) = d2Udmm(:,:,j,i)*dEdt(:,:,i);
119         d2Udmm(:,:,j,i+nft) = d2Udmm(:,:,j,i+nft)*dEdf(:,:,i);
120         d2Udmm(:,:,j+nft,i+nft) = d2Udmm(:,:,j+nft,i+nft)*dEdf(:,:,i);
121         for k = i+1:nft
122             d2Udmm(:,:,j,i) = d2Udmm(:,:,j,i)*E(:,:,k);
123             d2Udmm(:,:,j,i+nft) = d2Udmm(:,:,j,i+nft)*E(:,:,k);
124             d2Udmm(:,:,j+nft,i+nft) = d2Udmm(:,:,j+nft,i+nft)*E(:,:,k);
125         end
126     end
127
128     d2Udmm(:,:,i+nft,i+nft) = d2Udmm(:,:,i,i)*d2Edff(:,:,i);  % wrt same phi
129     d2Udmm(:,:,i,i+nft) = d2Udmm(:,:,i,i)*d2Edft(:,:,i);  % wrt theta and phi of
    same index
130     d2Udmm(:,:,i,i) = d2Udmm(:,:,i,i)*d2Edtt(:,:,i);  % wrt same theta
131
132     for j = i+1:nft
133         d2Udmm(:,:,j,i+nft) = eye(d);
134         for k = 1:i-1
135             d2Udmm(:,:,j,i+nft) = d2Udmm(:,:,j,i+nft)*E(:,:,k);
136         end
137         d2Udmm(:,:,j,i+nft) = d2Udmm(:,:,j,i+nft)*dEdf(:,:,i);
138         for k = i+1:j-1
139             d2Udmm(:,:,j,i+nft) = d2Udmm(:,:,j,i+nft)*E(:,:,k);
140         end
141         d2Udmm(:,:,j,i+nft) = d2Udmm(:,:,j,i+nft)*dEdt(:,:,j);
142         for k = j+1:nft
143             d2Udmm(:,:,j,i+nft) = d2Udmm(:,:,j,i+nft)*E(:,:,k);
144         end
145         d2Udmm(:,:,i,i) = d2Udmm(:,:,i,i)*E(:,:,j);  % wrt same theta
146         d2Udmm(:,:,i,i+nft) = d2Udmm(:,:,i,i+nft)*E(:,:,j);  % wrt theta and phi
    of same index
147         d2Udmm(:,:,i+nft,i+nft) = d2Udmm(:,:,i+nft,i+nft)*E(:,:,j);  % wrt same
    phi
148     end
149 end
150
151 % matrix of alpha
152 % off diagonal elements are zero
153 D = zeros(d,d);
154 for i = 1:d-1
155     D(i,i) = (cosa(i))^2;
156     for j = 1:i-1
157         D(i,i) = D(i,i)*(sina(j))^2;
158     end
159 end
```

```
160 D(d,d) = D(d-1,d-1)*(tana(d-1))^2;
161
162 rho = cU*D*U;
163
164 dDda = zeros(d,d,na);        % 1st order derivative of D wrt alpha
165 d2Ddaa = zeros(d,d,na,na);  % 2nd order derivative of D wrt alpha
166 for i = 1:na
167     dDda(i,i,i) = -2*D(i,i)*tana(i);
168     d2Ddaa(i,i,i,i) = -2*D(i,i)*(1-(tana(i))^2);
169     for j = 1:i-1
170         d2Ddaa(i,i,j,i) = -4*D(i,i)*tana(i)*cota(j);
171         for k = i+1:d
172             d2Ddaa(k,k,j,i) = 4*D(k,k)*cota(i)*cota(j);
173         end
174     end
175     for j = i+1:d
176         dDda(j,j,i) = 2*D(j,j)*cota(i);
177         d2Ddaa(j,j,i,i) = 2*D(j,j)*((cota(i))^2-1);
178     end
179 end
180
181 jcb = zeros(d^2-1);
182 for i = 1:d^2-1
183     for j = 1:nft+nft  % wrt theta and phi
184         jcb(i,j) = 2*real(trace(dUdm(:,:,j)*D*cU*Q(:,:,i)));
185     end
186     for j = 1:na  % wrt alpha
187         jcb(i,j+nft+nft) = trace(U*dDda(:,:,j)*cU*Q(:,:,i));
188     end
189 end
190
191 % determinant of Jacobian matrix dpdm
192 JacDet = det(jcb);
193
194 M = zeros(d^2-1,d^2-1,d^2-1);
195 for i = 1:nft+nft     % wrt theta and phi
196     for j = 1:d^2-1
197         for k = 1:nft+nft
198             M(j,k,i) = 2*real(trace((d2Udmm(:,:,min(i,k),max(i,k))*D*cU+dUdm(:,:,
    k)*D*cdUdm(:,:,i))*Q(:,:,j)));
199         end
200         for k = 1:na
201             M(j,k+nft+nft,i) = 2*real(trace(dUdm(:,:,i)*dDda(:,:,k)*cU*Q(:,:,j)))
    ;
202         end
203     end
204 end
205 for i = 1:na  % wrt alpha
206     for j = 1:d^2-1
207         for k = 1:nft+nft
208             M(j,k,i+nft+nft) = M(j,i+nft+nft,k);
209         end
210         for k = 1:na
211             M(j,k+nft+nft,i+nft+nft) = trace(U*d2Ddaa(:,:,min(i,k),max(i,k))*cU*Q
    (:,:,j));
```

```
212         end
213     end
214 end
215
216 u = zeros(1,d^2-1);
217 inv_jcb = eye(d^2-1)/jcb; % inv_jcb = inv(jcb);
218 for i = 1:d^2-1
219     u(i) = real(trace(inv_jcb*M(:,:,i)));  % inverse of jcb * M(:,:,i)
220 end
```

A.4 Fidelity and Distance

```
% input data points
% output fidelity histogram, distance histogram, and fidelity vs. distance

clear all
close all
warning('off','all');

% choose one workspace to load based on your prior density

load('hmc_AA_2qb_primitive_1m_pts.mat');
% load('hmc_AA_2qb_Jeffreys_1m_pts.mat');
% load('hmc_AA_2qb_hedged_1m_pts.mat');

ln = length(rho);
randInd = zeros(1,2);
randMat = zeros(4);
fdl = zeros(1,ln);  % fidelity
dist = zeros(1,ln); % distance
dx = 100;  % number of bins
x_axis = 1/dx:1/dx:1;  % x axis of distribution plot
for i = 1:ln
    % random number between 1 and length of rho
        randInd = round(1+(ln-1).*rand(1,2));
    % randomly choose two matrices and multiply them together
    randMat = rho(:,:,randInd(1))*rho(:,:,randInd(2));

    randEig = sqrt(eig(randMat));  % square root of eigenvalues
    fdl(i) = real(sum(randEig));   % sum up to get fidelity

    randMat2 = rho(:,:,randInd(1))-rho(:,:,randInd(2));
    randMat2 = randMat2'*randMat2;
    dist(i) = trace(sqrtm(randMat2))/2; % trace distance
end

h = figure;
[f,x] = hist(fdl,dx);
bar(x_axis,f/ln);  %  fidelity distribution
axis([0,1,0,0.04]); xlabel('Fidelity');ylabel('Prior Density');
% title(['Fidelity Distribution (', priorType, ', 1m points)']);
fileName1 = strcat(fileName, '_fidelity');
set(gca,'xlim',[0 1]); set(gca,'FontSize',14);
print(h, '-djpeg', fileName1);

h = figure;
[d,x] = hist(dist,100);
bar(x_axis,d/ln);  %  distance distribution
axis([0,1,0,0.04]); xlabel('Distance');ylabel('Prior Density');
% title(['Distance Distribution (', priorType, ', 1m points)']);
fileName2 = strcat(fileName, '_distance');
set(gca,'xlim',[0 1]); set(gca,'FontSize',14);
print(h, '-djpeg', fileName2);

h = figure;
```

```
54 scatter(dist,fdl,'filled');  % fidelity vs. distance
55 axis([0,1,0,1]);xlabel('Distance');ylabel('Fidelity');
56 % title(['Fidelity vs. Distance (', priorType, ', 1m points)']);
57 fileName3 = strcat(fileName, '_fidelity_vs_distance');
58 set(gca,'FontSize',14);
59 print(h, '-djpeg', fileName3);
60
61 h = figure;
62 hist3([dist;fdl].',[50,50]);  % 3d histogram of fidelity and distance
63 xlabel('Distance'); ylabel('Fidelity');
64 % title(['Fidelity and Distance (', priorType, ', 1m points)']);
65 set(get(gca,'child'),'FaceColor','interp','CDataMode','auto');
66 set(gca,'FontSize',14);
67 fileName4 = strcat(fileName, '_fidelity_distance_3d_histogram');
68 print(h, '-djpeg', fileName4);
69 % fileName5 = strcat(fileName, '_fidelity_distance_2d');
70 % view(0,90);
71 % print(h, '-djpeg', fileName5);
```

Bibliography

[1] Jiangwei Shang, Hui Khoon Ng, Arun Sehrawat, Xikun Li, and Berthold-Georg Englert. Optimal error regions for quantum state estimation. *New Journal of Physics*, 15(12):123026, 2013.

[2] Jiangwei Shang, Yi-Lin Seah, Hui Khoon Ng, David John Nott, and Berthold-Georg Englert. Monte carlo integration over regions in the quantum state space. i. *arXiv preprint arXiv:1407.7805*, 2014.

[3] Yi-Lin Seah, Jiangwei Shang, Hui Khoon Ng, David John Nott, and Berthold-Georg Englert. Monte carlo integration over regions in the quantum state space. ii. *arXiv preprint arXiv:1407.7806*, 2014.

[4] Jaroslav Řeháček, Berthold-Georg Englert, and Dagomir Kaszlikowski. Minimal qubit tomography. *Physical Review A*, 70(5):052321, 2004.

[5] Asher Peres. Separability criterion for density matrices. *Physical Review Letters*, 77 (8):1413, 1996.

[6] Karol Życzkowski, Paweł Horodecki, Anna Sanpera, and Maciej Lewenstein. Volume of the set of separable states. *Physical Review A*, 58(2):883, 1998.

[7] Christopher A Fuchs and Jeroen Van De Graaf. Cryptographic distinguishability measures for quantum-mechanical states. *Information Theory, IEEE Transactions on*, 45(4):1216–1227, 1999.