Computer Physics Communications 183 (2012) 118–124

Contents lists available at ScienceDirect

Computer Physics Communications

www.elsevier.com/locate/cpc

Generating and using truly random quantum states in *Mathematica* ☆

Jarosław Adam Miszczak

Institute of Theoretical and Applied Informatics, Polish Academy of Sciences, Bałtycka 5, 44-100 Gliwice, Poland

ARTICLE INFO

Article history:
Received 16 March 2011
Received in revised form 30 July 2011
Accepted 5 August 2011
Available online 18 August 2011

Keywords:
Random density matrices
Quantum information
Quantum random number generators

ABSTRACT

The problem of generating random quantum states is of a great interest from the quantum information theory point of view. In this paper we present a package for *Mathematica* computing system harnessing a specific piece of hardware, namely Quantis quantum random number generator (QRNG), for investigating statistical properties of quantum states. The described package implements a number of functions for generating random states, which use Quantis QRNG as a source of randomness. It also provides procedures which can be used in simulations not related directly to quantum information processing.

Program summary

Program title: TRQS
Catalogue identifier: AEKA_v1_0
Program summary URL: http://cpc.cs.qub.ac.uk/summaries/AEKA_v1_0.html
Program obtainable from: CPC Program Library, Queen's University, Belfast, N. Ireland
Licensing provisions: Standard CPC licence, http://cpc.cs.qub.ac.uk/licence/licence.html
No. of lines in distributed program, including test data, etc.: 7924
No. of bytes in distributed program, including test data, etc.: 88 651
Distribution format: tar.gz
Programming language: Mathematica, C
Computer: Requires a Quantis quantum random number generator (QRNG, http://www.idquantique.com/true-random-number-generator/products-overview.html) and supporting a recent version of Mathematica
Operating system: Any platform supporting Mathematica; tested with GNU/Linux (32 and 64 bit)
RAM: Case dependent
Classification: 4.15
Nature of problem: Generation of random density matrices.
Solution method: Use of a physical quantum random number generator.
Running time: Generating 100 random numbers takes about 1 second, generating 1000 random density matrices takes more than a minute.

1. Introduction

As full scale quantum computing devices are still missing, the simulation of quantum computers has gained considerable attention as a method for investigation the behaviour of quantum algorithms and protocols [1]. It also provides a valuable method for inspecting the mathematical structure of quantum theory by providing information about statistical properties of quantum states and operations [2].

Generating random numbers using the statistical nature of quantum theory provides one of the first practical applications of quantum information theory. At the same time the high quality of random numbers generated using quantum random number generators is based on the very basic principles of Nature [3]. Since random numbers are important in many areas of human activity, at the moment this provides one of the most important applications of quantum information theory.

During the last several years many simulators of quantum computers have been developed and the most up-to-date list of available software is available at [1]. Many simulators of quantum information processing were developed using *Mathematica* computing system [4–8]. Also some attention has been devoted to utilising CUDA programming model [9] and parallel processing model [10–12]. The research effort in using parallel and distributed computing for the purpose of quantum information processing was motivated by the amount of computational resources needed in order to perform a simulation of quantum computation.

☆ This paper and its associated computer program are available via the Computer Physics Communications homepage on ScienceDirect (http://www.sciencedirect.com/science/journal/00104655).

E-mail address: miszczak@iitis.pl.

0010-4655/$ – see front matter
doi:10.1016/j.cpc.2011.08.002

In this paper we present a **TRQS** (True Quantum Random States) package for *Mathematica* computing system harnessing a specific piece of hardware, namely Quantis quantum random number generator (QRNG), for investigating statistical properties of quantum states. The described package implements a number of functions for using numbers and generating random states (i.e. random state vectors and random density matrices), using Quantis QRNG as the source of randomness. The presented package has been developed for the purpose of quantum information theory, but it can be easily utilised in other areas of science.

The motivation for utilising random numbers generated using quantum devices is twofold. Firstly, QRNGs provide a high-quality source of randomness which can be used in various areas of computational physics. Recent progress in this area [13] suggests that QRNGs are of a great interest for experimentalists, as well as theorists, working in the field of quantum information theory. Secondly, it has been shown that the statistical properties of obtained numbers cannot be reproduced using standard methods of generating random numbers [3].

This paper is organised as follows. In Section 2 we introduce basic theoretical facts concerning random states and operations. In Section 3 we describe the functions implemented in the presented package and in Section 4 we use the described package to analyse some problems related to quantum information theory. We also use **TRQS** package be benchmark the speed of Quantis random number generator. Finally, in Section 5, we provide some concluding remarks and discuss the alternative sources of random numbers generated using quantum random number generators.

2. Random quantum states

The problem of generating random quantum states is of a great interest from the quantum information theory point of view. Random states appear naturally in many situations in quantum information processing, especially when one must deal with the unavoidable interaction of the system in question with the environment.

We start by recalling some basic facts used in the rest of this paper. For a more complete introduction to mathematical concepts used in quantum information theory see e.g. [14,2]. Next, we present the selected methods of generating random density matrices implemented in the **TRQS** package. More detailed description of the methods for generating random quantum density matrices can be found in [15].

2.1. Basic definitions

In what follows we restrict our attention to finite-dimensional spaces. We denote by $|\phi\rangle \in \mathbb{C}^n$ *pure states* i.e. normalised elements of the vector space $\mathbb{C}^n$. By $\mathbb{M}_{m,n}$ we denote the set of all $m \times n$ matrices over $\mathbb{C}$ and the set of square $n \times n$ matrices is denoted by $\mathbb{M}_n$. The set of n-dimensional density matrices (normalised, positive semi-definite operators on $\mathbb{C}^n$) is denoted by Ω_n. The set $\mathbb{M}_n$ has the structure of a Hilbert space with the scalar product given by $(A, B) = \operatorname{tr} A^\dagger B$. This particular Hilbert space is known as the Hilbert–Schmidt space of operators acting on $\mathbb{C}^n$ and we will denote it by $\mathcal{H}_{\mathrm{HS}}$.

In particular, $\Omega_n \subset \mathbb{M}_n$. Moreover, any element of Ω_n can be represented as a convex combination (mixture) of one-dimensional projectors. For any $\rho \in \Omega_n$ there exists a sequence of non-negative numbers $p_1, p_2, \ldots, p_n$ such that $\rho = \sum_{i=1}^n p_i P_i$, where $\{P_i\}$, $i = 1, 2, \ldots, n$, is a sequence of orthonormal one-dimensional projectors. Extreme elements of the set Ω_n are exactly the *pure states* and can be identified with ket vectors $|\psi\rangle \simeq |\psi\rangle\langle\psi|$. Convex combinations of pure states are refereed to as *mixed states*.

2.2. Random pure states

In most cases to describe quantum algorithms and protocols one assumes that it is possible to avoid unnecessary interactions with the environment [2]. In such situation the state of the system remains pure during the evolution, which is represented by a unitary matrix.

In the case of a pure state (state vectors) there exists a natural measure in the set, namely the measure generated by the Haar measure on the group of unitary matrices $\mathrm{U}(n)$. The algorithm for generating random pure states is presented in Procedure 1. The function **RandomSimplex(n)** used in this procedure returns an element of a standard simplex of dimension n.

Procedure 1 Generation of a random pure state.

Input: $n \geqslant 0$
Output: Random pure state v of dimension n
  $s \leftarrow$ **RandomSimplex(n)**
  $a[1] \leftarrow \sqrt{s[1]}$
  $p[1] \leftarrow 1$
  $v[1] \leftarrow a[1] * p[1]$
  for $k = 2$ **to** n **do**
    $a[k] \leftarrow \sqrt{s[i]}$
    $p[k] \leftarrow \exp(i\,$**RandomReal**$(0, 2\pi))$
    $v[k] \leftarrow a[k] * p[k]$
  end for
  return v

Alternatively a random pure states can be obtained by generating a random unitary matrix and choosing its columns as random pure states.

2.3. Random mixed states

The need for using a more general formalism to describe the evolution of quantum systems is motivated by the fact that in a real-world situation it is impossible to avoid the interaction of the system with the environment. In this case one needs to represent the system using quantum channels and introduce density matrices to describe the state of the system [2].

The set of density matrices presents us with more complicated structure than in the case of pure states. In particular, it is not possible to distinguish one preferred probability measure in this set and any metric on the set can be used to introduce one.

The package presented in this paper implements functions for generating random density matrices distributed according to the probability measure generated by the Hilbert–Schmidt metric

$$\|\rho_1 - \rho_2\|_{\mathrm{HS}} = \sqrt{\operatorname{tr}\big[(\rho_1 - \rho_2)^2\big]}, \tag{1}$$

and the Bures metric

$$\|\rho_1 - \rho_2\|_{\mathrm{B}} = \sqrt{2 - 2\sqrt{F(\rho_1, \rho_2)}} \tag{2}$$

where $F(\rho_1, \rho_2)$ is a quantum fidelity $F(\rho_1, \rho_2) = \operatorname{tr}|\sqrt{\rho_1}\sqrt{\rho_2}|$ between two density matrices. In a particular case, when one of the states is pure, $\rho_1 = |\psi\rangle\langle\psi|$, we have $F(|\phi\rangle\langle\phi|, \rho_2) = \langle\phi|\rho_2|\psi\rangle$ and in this case the probability measure is reduced to the Fubini–Study measure.

We also provide a function for generating density matrices distributed with a family of induced measures [2], which can be derived by averaging over an external subsystem. One should note that the Hilbert–Schmidt measure can be obtained as an induced measure.

In each case, as a starting point of the algorithm, one needs to use a Ginibre matrix, i.e. a complex matrix with elements having real and complex parts distributed with the normal distribution $\mathcal{N}(x, y)$ [2].

Procedure 2 Generation of the random matrix from the Ginibre ensemble.

Input: $m, n > 0$
Output: Matrix G of size $m \times n$
  for $k = 1$ **to** m **do**
    for $l = 1$ **to** n **do**
      $G[k, l] \leftarrow$ **RandomReal**$(0, 1) + i$ **RandomReal**$(0, 1)$
    end for
  end for
  return G

Using *Mathematica* language, Procedure 2 can be written in a compact form as

```
dist = NormalDistribution[0,1];
GinibreMatrix[m_,n_]:=
    RandomReal[dist,{m,n}]
    + I RandomReal[dist,{m,n}]
```

2.3.1. Induced measures and the Hilbert–Schmidt ensemble

The Hilbert–Schmidt metric defined in Eq. (1) is commonly used to describe the metric structure of the set of quantum states. This distance introduces a Euclidean geometry in the space of density matrices. In the special case of one-qubit density matrices, the space has the form of the Bloch ball.

The Hilbert–Schmidt measure belongs to the class of *induced measures* [2, Ch. 14]. In a general case, one can seek a source of randomness in a given system, by studying the interaction of the n-dimensional system in question with the environment. In such situation the random states to model the behaviour of the system should be generated by reducing a pure state in $N \times K$-dimensional space. In what follows we denote the resulting probability measure by $\mu_{N,K}$.

In particular, the Hilbert–Schmidt probability measure on n-dimensional space Ω_n can be obtained by reducing a bi-partite pure quantum state from $\mathbb{C}^{N\times N}$. However, it is easy to see that this measure, as well as any measure $\mu_{N,K}$, can be obtained by using a simpler procedure.

We start by observing that any complex matrix $X \in \mathbb{M}(\mathbb{C})$ can be used to construct a normalised, positive matrix $\frac{XX^\dagger}{\operatorname{tr} XX^\dagger}$. Let us now assume that we have a pure state in the $N \times K$-dimensional Hilbert space, $|\psi\rangle \in \mathcal{H}_N \otimes \mathcal{H}_K = \mathbb{C}^{N\times K}$. Any such state can be represented in a product basis $|\psi\rangle = \sum_{i=1}^{N} \sum_{j=1}^{K} X_{ij} |i\rangle \otimes |j\rangle$, where $X \in \mathbb{M}_{N,K}$. The matrix $XX^\dagger$ is, in this case, equivalent to the partial trace of $|\psi\rangle\langle\psi|$ with respect to the second subsystem, $\operatorname{tr}_{\mathcal{H}_K} |\psi\rangle\langle\psi| = XX^\dagger$. Symmetrically we have $\operatorname{tr}_{\mathcal{H}_N} |\psi\rangle\langle\psi| = X^\dagger X$.

It follows from the above considerations that the spectrum of a density matrix obtained using this method is the set of squared and normalised singular values of the matrix X. In particular, if one assumes that the pure states on $\mathcal{H}_K \otimes \mathcal{H}_N$ are distributed according to the Fubini–Study measure, the resulting density matrices are distributed with induced measures.

In the case of induced measures the elements of the coefficient matrix are independent random variables and form a Ginibre matrix.

Procedure 3 Generation of a random density matrix distributed according to induced probability measure $\mu_{n,k}$ obtained by tracing out the ancillary system of dimension k.

Input: $n \geqslant 0, k \geqslant 2$
Output: Random mixed state ρ of dimension n
  $G \leftarrow$ **GinibreMatrix(n, k)**
  $\rho \leftarrow GG^\dagger$
  $\rho \leftarrow \frac{1}{\operatorname{tr}\rho}\rho$
  return ρ

In the special case of $K = N$ we obtain the Hilbert–Schmidt ensemble.

The statistical properties of the set of quantum states with respect to the probability measure introduced by the Hilbert–Schmidt metric were studied in [16,17].

2.3.2. Bures ensemble

Another popular measure of the distance between quantum states is the Bures distance. Its usage is motivated by the fact that this distance, when restricted to diagonal matrices, is equivalent to the Hellinger distance in statistics, defined for two discrete probability distributions as $H(\mathbf{p}, \mathbf{q}) = \sum_i \sqrt{p_i q_i}$. Moreover, the Bures distance and quantum fidelity are related to the distinguishability of quantum states, defined as a trace distance between the given states.

The above features distinguish the Bures measure as an optimal method for generating random density matrices in the situation when no information about the source of state is present.

The algorithm for generating random density matrices distributed according to the probability measure based on the Bures distance was provided in [18]. This algorithm is presented in Procedure 4.

Procedure 4 Generation of a random density matrix distributed according to the probability measure induced by the Bures metric.

Input: $n \geqslant 0$
Output: Random mixed state ρ of dimension n
  $G \leftarrow$ **GinibreMatrix(n, n)**
  $U \leftarrow$ **RandomUnitary(n)**
  $\rho \leftarrow (1 + U)GG^\dagger(1 + U^\dagger)$
  $\rho \leftarrow \frac{1}{\operatorname{tr}\rho}\rho$
  return ρ

3. Description of the package

The **TRQS** package implements a number of functions allowing to obtain random state vectors and random density matrices.

It uses Quantis random number generator produced by ID Quantique [19] for the purpose of generating random numbers. ID Quantique provides drivers, example programs and a library for accessing Quantis devices on most popular operating systems. The software package for using Quantis, including some examples of how `libQuantis` library can be used, can be downloaded from the ID Quantique support page [20].

Note that the functions implemented are independent from the actual source of randomness. In particular it is possible to switch to some other sources of random numbers.

The package consists of a set of source files, developed using *MathLink* provided by *Mathematica* and a package file `TRQS.m`, implementing the main functionality.

3.1. Communication with Quantis device

In the presented package the communication between *Mathematica* and Quantis device was implemented using *MathLink* – a standard interface for interprogramme communication provided by *Mathematica*.

We used `libQuantis` library that provides a number of functions for reading random data from Quantis device. In particular the presented package uses only functions for reading basic data types. The functions of this kind implemented in `libQuantis` can be divided into four categories:

- `QuantisReadShort` and `QuantisReadScaledShort` – functions for reading short integers and short integers within the given range,

- `QuantisReadInt` and `QuantisReadScaledInt` – functions for reading long integers and long integers within the given range,
- `QuantisReadFloat_01` and `QuantisReadScaledFloat` functions for reading float numbers in the range $[0, 1]$ and float numbers within the given range,
- `QuantisReadDouble_01` and `QuantisReadScaledDouble` – functions for reading double numbers in the range $[0, 1]$ and double numbers within the given range.

Moreover, the library function `QuantisRead` allows to read raw random data. This function can be also used to read large amount of data, which can be necessary in the case when one needs to fill large matrices with random numbers.

3.2. Organisation of the package

The described package was designed to work in companion with the QI package for *Mathematica* [7] and can be used along with this package. The provided functions for generating random states can be grouped into three categories: basic functions, functions for generating pure states and unitary matrices and functions for generating mixed states and channels.

The functions are defined within the **TRQS** name space. We follow the naming convention which assumes that functions using a true random number generator to produce results have names starting with **True** and the rest of the name describes the generated object. The functions related to the configuration of the back-end i.e. Quantis device, have names starting with **Quantis** (see: Section 3.2.4).

Each function is provided along with some basic information about its functionality.

3.2.1. Basic functions

The first group of functions implements basic structures utilised for generating quantum states. The functions in this group implement communication with Quantis device and provide the generation of real and integer random numbers and some basic structures. In this group the following functions allow to access basic types of random numbers:

1. `TrueRandomReal` – returns a random real (double) number; this function is based on `libQuantis` library function `QuantumReadScaledDouble` and is implemented in three variants:
   (a) `TrueRandomReal[`$\{n_{\min}, n_{\max}\}$`]` – returns a real number distributed uniformly in the interval $[n_{\min}, n_{\max}]$,
   (b) `TrueRandomReal[`$n_{\max}$`]` – returns a real number distributed uniformly in the interval $[0, n_{\max}]$,
   (c) `TrueRandomReal[]` – returns a real number distributed uniformly in the interval $[0, 1]$.
2. `TrueRandomRealNormal[x,y,`$\{d_1, \ldots, d_l\}$`]` – returns a $(d_1 \times \cdots \times d_l)$-dimensional array of random numbers distributed according to $\mathcal{N}(x, y)$.
3. `TrueRandomInteger` – returns a random integer. This function, based on `libQuantis` library function `QuantumReadScaledInt`, is provided for convenience and implemented in three variants:
   (a) `TrueRandomInteger[`$\{n_{\min}, n_{\max}\}$`]` – returns an integer distributed uniformly in the interval $[n_{\min}, n_{\max}]$,
   (b) `TrueRandomInteger[`$n_{\max}$`]` – returns an integer distributed uniformly in the interval $[0, n_{\max}]$,
   (c) `TrueRandomInteger[]` – returns 0 or 1.

The following functions, built using these basic functions, allow to obtain the structures used to construct random quantum states and operations:

1. `TrueRandomSimplex[n]` – returns an element of a standard simplex, distributed uniformly on simplex,
2. `TrueGinibreMatrix[m,n]` – returns an $n \times m$ Ginibre matrix,
3. `TrueRandomChoice[`$\{e_1, e_2, \ldots, e_n\}$`]` – returns at random one of the $\{e_1, e_2, \ldots, e_n\}$,
4. `TrueRandomGraph[v,e, form]` – returns a pseudo-random graph with v vertices and e edges. Additionally, the last argument can be set to “Graph” (default) to obtain a graphical representation of the result or to “List” to obtain the result as a list of vertices and edges.

3.2.2. Pure states and unitary matrices

The functions in this group allow to obtain random pure states and random unitary matrices. Since product (or local) states and operations are of a special interest in quantum information theory, we provide functions allowing to generate pure states and unitary matrices of the tensor product structure:

1. `TrueRandomKet[n]` – returns a random pure state in n-dimensional space $\mathbb{C}^n$,
2. `TrueRandomProductKet[`$\{n_1, n_2, \ldots, n_k\}$`]` – returns a random pure state, which is an element of space with the tensor product structure $\mathbb{C}^{n_1} \otimes \mathbb{C}^{n_2} \otimes \cdots \otimes \mathbb{C}^{n_k}$
3. `TrueRandomUnitary[n]` – returns a random unitary matrix acting on n-dimensional space $\mathbb{C}^n$,
4. `TrueRandomLocalUnitary[`$\{n_1, n_2, \ldots n_k\}$`]` – returns a random unitary matrix, which acts on the elements of space with the tensor product structure $\mathbb{C}^{n_1} \otimes \mathbb{C}^{n_2} \otimes \cdots \otimes \mathbb{C}^{n_k}$.

3.2.3. Mixed states

The last group of functions implements the generation of random mixed states. In particular we have:

1. `TrueRandomStateHS[n]` – a random density matrix of dimension n, generated according to the Hilbert–Schmidt measure,
2. `TrueRandomStateBures[n]` – a random density matrix of dimension n, generated according to the Bures measure,
3. `TrueRandomStateInduced[n,k]` – a random density matrix of dimension n, generated according to the induced probability measure with an external system of dimension k,
4. `TrueRandomProductState[`$\{n_1, n_2, \ldots, n_k\}, \mu$`]` – a product random density matrix acting on the space with the tensor product structure $\mathbb{C}^{n_1} \otimes \mathbb{C}^{n_2} \otimes \cdots \otimes \mathbb{C}^{n_k}$ and with each local component generated according to measure μ, where μ can be set to `"HS"`, `"Bures"` or some integer K describing an induced measure.

Additionally **TRQS** package allows to generate random dynamical matrices, representing the most general form of quantum system evolution.

1. `TrueRandomDynamicalMatrix[n,k]` – a random dynamical matrix of dimension n, representing a quantum channel acting on n-dimensional space of density matrices, with k eigenvalues set to 0. The last argument is set to 0 by default.

The above function is based on the algorithm described in [21]. The obtained random dynamical matrix can be easily transformed into a set of random Kraus operators [2,22].

3.2.4. Functions related to the back-end configuration

To provide some basic interaction with the underlying device, the following functions were implemented in **TRQS** package.

1. `QuantisGetLibVersion[]` – returns a version number of the installed libQuantis library.
2. `QuantisGetSerialNumber[]` – returns a serial number of Quantis device used as a back-end.
3. `QuantisGetDeviceID[]` – returns an id number of Quantis device.
4. `QuantisGetDeviceType[]` – returns a type of Quantis device.

Note that the functions `QuantisGetDeviceID[]` and `QuantisGetDeviceType[]` provide only information about the configuration options used during the compilation of *MathLink* source files.

4. Examples

The main aim of the presented package is to provide a tool for the analysis of the properties of random density matrices. Below we present two examples of such analysis. First, we calculate the distributions of eigenvalues for 4-dimensional mixed density matrices and compare analytical and numerical results. Next, we calculate numerically the average fidelity between random density matrices with respect to measure $\mu_{2,K}$. In both cases we compare the results obtained using the presented package and the results obtained from a standard random number generator with the analytical results.

We also provide a comparison of speed between the standard pseudo-random number generator from *Mathematica* and generator using `libQuantis` library. This example shows that the speed of random number generation offered by the currently available hardware is insufficient.

4.1. Distribution of eigenvalues

The Bures and Hilbert–Schmidt probability measures are of the product form i.e. the distribution of eigenvalues is independent from the distribution of eigenvectors.

In the case of the Hilbert–Schmidt measure the probability density of eigenvalues is given by the formula [2]

$$P_{\mathrm{HS}}(\lambda_1,\dots,\lambda_N)=C_N^{\mathrm{HS}}\prod_{i<j}(\lambda_i-\lambda_j)^2, \tag{3}$$

where $\sum_i \lambda_i = 1,\ \lambda_i \leqslant 0, i = 1, 2, \dots, N$. The normalisation constant C_N^{HS} reads

$$C_N^{\mathrm{HS}}=\frac{\Gamma(N^2)}{\prod_{i=1}^{N}\Gamma(k)\Gamma(k+1)}. \tag{4}$$

Here we present some results for the Hilbert–Schmidt measure and density matrices of dimension 4. The distribution of eigenvalues $\lambda_1, \lambda_2, \lambda_3, \lambda_4$ of the random density matrices from Ω_4 generated uniformly with respect to the Hilbert–Schmidt measure is presented in Fig. 1(a). In Fig. 1(b) the distribution of $\lambda_1, \lambda_2, \lambda_3, \lambda_4$ obtained using true random density matrices is presented. Numerical results were obtained using a sample of 2000 random density matrices.

4.2. Average fidelity

Quantum fidelity [2] is commonly used in quantum information theory to quantify to what degree a given quantum state can be approximated by some other state or a family of states [23].

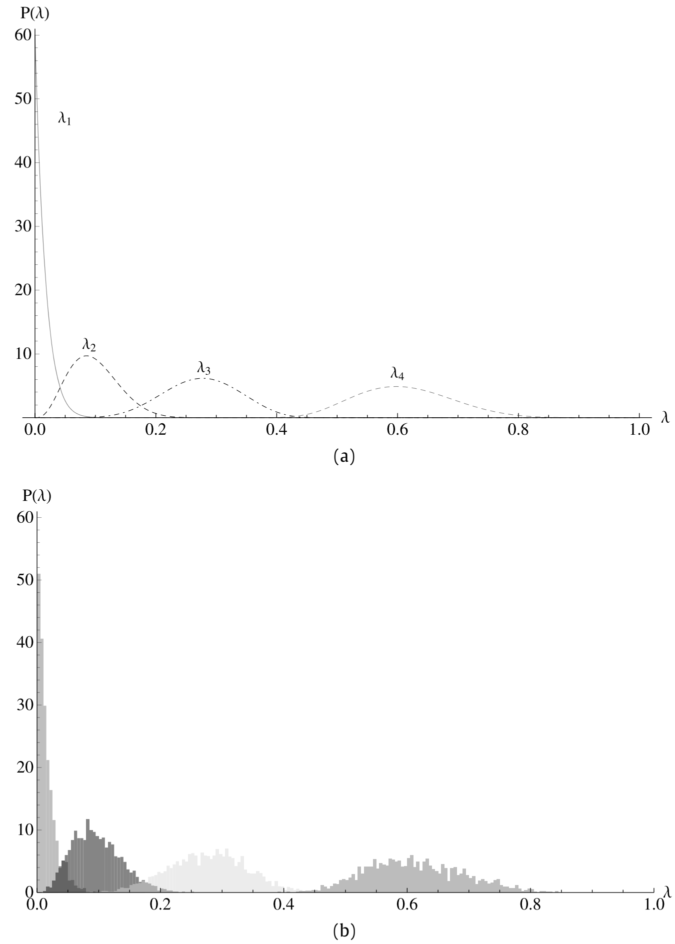

Fig. 1. Distribution of eigenvalues for random density matrices distributed uniformly according to the Hilbert–Schmidt probability measure. (a) Analytical results obtained by a direct integration over appropriate subsets of the convex hull of the spectrum. (b) Numerical results obtained using random states generated with TRQS package.

The average fidelity between two random quantum states can be used e.g. to provide an insight into the performance of quantum protocols in the presence of noise. Since, in most cases, in quantum information processing one is interested in the behaviour of 2-dimensional systems (qubits), below we deal with this case only.

As it has already been mentioned, the use of random states in quantum information processing is commonly motivated by the interaction of the system in question with the environment. In this case one is interested in random density matrices generated uniformly with respect to some induced measure $\mu_{2,K}$, where K is the dimension of the ancillary system.

The mean fidelity between two one-qubit random density matrices generated uniformly with respect to measure $\mu_{N,K}$ was calculated in [17] and reads

$$\langle F\rangle_{2,K}=\frac{1}{2}+\frac{1}{2}\left(\frac{\Gamma(K-\frac{1}{2})\Gamma(K+\frac{1}{2})}{\Gamma(K-1)\Gamma(K+1)}\right)^2. \tag{5}$$

The average fidelity for one-qubit random states generated with $\mu_{2,K}$ is presented in Fig. 2. The results were obtained using a sample of 50 states and one can see that in this case it allows to obtain a very good approximation of an exact result, especially in the case of large K.

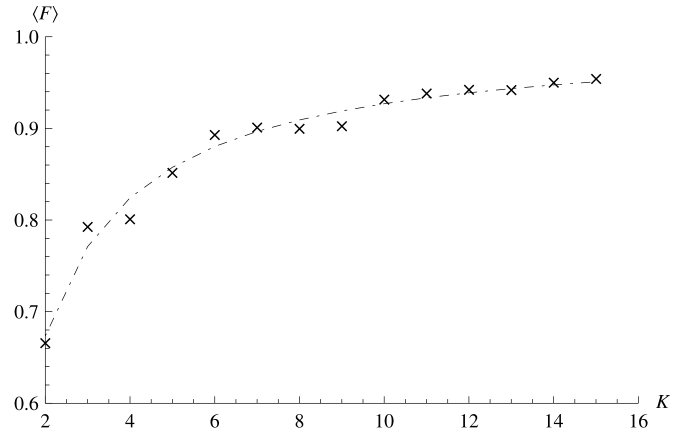

Fig. 2. Average fidelity between one-qubit random mixed states generated uniformly with respect to $\mu_{2,K}$. The dotted line represents the exact result. Numerical results obtained using the presented package are marked with "$\times$".

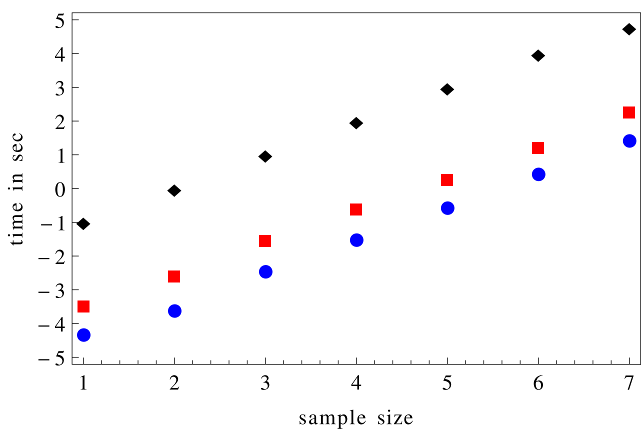

Fig. 3. Comparison of speed for random number generators in log–log scale. Blue circles represent timings for samples generated using `RandomReal[]` functions using pseudo-random number generator. Red squares represent timings for samples generated with **TRQS** package function `TrueRandomReal[]` using *MathLink* executables linked against `libQuantis-NoHW` library. Black diamonds illustrate timings for an analogous method with *MathLink* executables linked against `libQuantis` library.

4.3. Speed comparison

For the purpose of testing the speed of the **TRQS** package we have performed three experiments involving generation of random real numbers distributed uniformly on the unit interval. In each experiment we have used a different method.

1. The first experiment was conducted using a standard pseudo-random number generator provided by *Mathematica*. Additionally we used `ClearSystemCache["Numeric"]` in order to generate the results independent from previous computations.
2. In the second experiment the numbers were generated using **TRQS** package and *MathLink* executables linked against `libQuantis-NoHW` library. While in this case the generated numbers are still pseudo-random, this test was included in order to measure the overhead stemed from the access to an external library.
3. The last experiment was conducted with **TRQS** package using data from the physical quantum random number generator.

In each experiment samples of size $10^1, 10^2, \ldots, 10^7$ were generated.

The obtained results are presented in Fig. 3. The comparison of timings for samples generated using pseudo-random number generator provided by *Mathematica* with timings for data obtained using Quantis device clearly shows that there is a tremendous difference in the speed of these generators. For example in order to generate a sample of 10^2 real numbers using a Quantis device one needs to wait about 1 s. Analogous sample is obtained in about 3×10^{-4} s when using a pseudo-random number generator provided by *Mathematica*.

At the same time, the sample of 10^2 can be obtained using **TRQS** package if the used *MathLink* executables are linked against `libQuantis-NoHW` library. This shows that the main overhead in generating random numbers using Quantis generator steams from the very slow physical scheme used to obtain random data.

5. Concluding remarks

Good random number generators are undoubtedly one of the most crucial elements used in computational physics. In particular, in simulations of quantum computing the use of random numbers is required to imitate the statistical behaviour of quantum mechanical objects, e.g. quantum register after measurement [24] or particle in quantum walks [25].

The described package can be used along with QI package for *Mathematica* [7] and some of the described functions are implemented in QI with the use of a pseudo-random number generator available in *Mathematica*. As the functions implemented in the presented package operate on basic data types available in *Mathematica*, it is also possible to use the package with other *Mathematica* packages developed for the simulation of quantum computing [5,6,8]. However, the potential application of the presented package is not limited to quantum information theory and the implemented functions can be used in other fields where good quality random numbers are required.

The obtained timings for different methods of producing random numbers suggest that the main obstacle in using the presented software in large scale simulations using random numbers is the speed of the random number generators. Clearly, at the moment the built-in pseudo-random number generator in *Mathematica* outperforms the Quantis-based random number generator. Quantis device provides a stream of random numbers generated at 4 Mbits/s. Additionally, the speed of random number generation is limited by the speed of the I/O operations. The speed of functions using Quantis QRNG can be improved by using `libQuantis` function **QuantisRead** for reading a larger amount of random data in the situation when e.g. large arrays are filled with random numbers. However, for the needs of simulations connected to quantum information theory, especially related to investigations of properties of low-dimensional systems the presented functions provide a satisfactory user experience. On the other hand, the recent progress in quantum random generation provides the methods for delivering random numbers generated at a rate of up to 50 Mbit/s [13] or higher [26].

Clearly the application of the described package is limited by the availability of Quantis quantum random number generator. However, alternative sources of random numbers generated using hardware operating on the basis of quantum mechanics exist. In particular, QRNG Service provided by PicoQuant GmbH and the Nano-Optics group at the Department of Physics of Humboldt University [27] allows to obtain samples of random numbers generated using quantum hardware. The samples can be downloaded directly via web page or, alternatively, using the provided library `libQRNG`. This library can be used in 32- and 64-bit versions of Linux and Windows operating systems. Another option is provided by the Quantum Random Bit Generator Service [28] developed by Centre for Informatics and Computing, Ruder Bošković

Institute, Zagreb, Croatia. This service provides bindings for a variety of programming languages, including C, Java and Python. Both services require registration and have some limitations concerning the amount of random data that can be downloaded. However, they provide free and relatively easy to use alternative for the commercial solution provided by ID Quantique.

Acknowledgements

Author would like to acknowledge interesting discussions with K. Życzkowski and W. Roga and some stimulating programming exercises conducted with Z. Puchała and P. Gawron. Thanks also goes to L. Widmer for motivating this work and H. Weier for his comments concerning the recent progress in the area of quantum random generation methods. Author would like to thank the anonymous reviewer for his motivating comments concerning the development of the package.

This work was supported by the Polish National Science Centre under the grant number N N516 475440 and by the Polish Ministry of Science and Higher Education under the grants number N N519 442339 and IP 2010 052 270.

References

[1] Quantiki, List of QC simulators, http://www.quantiki.org/wiki/List_of_QC_simulators.
[2] I. Bengtsson, K. Życzkowski, Geometry of Quantum States, Cambridge University Press, Cambridge, UK, 2006.
[3] C.S. Calude, M.J. Dinneen, M. Dumitrescu, K. Svozil, Experimental evidence of quantum randomness incomputability, Phys. Rev. A 82 (2) (2010) 022102.
[4] H. Touchette, P. Dumais, The quantum computation package for Mathematica 4.0, http://crypto.cs.mcgill.ca/QuCalc/.
[5] B. Juliá-Díaz, J.M. Burdis, F. Tabakin, QDENSITY—a Mathematica quantum computer simulation, Comp. Phys. Comm. 174 (2006) 914–934, http://www.pitt.edu/~tabakin/QDENSITY/QDENSITY.htm.
[6] J.L. Gómez-Muñoz, F. Delgado-Cepeda, Quantum 2.0 for Mathematica 7, http://homepage.cem.itesm.mx/lgomez/quantum/.
[7] J.A. Miszczak, Z. Puchała, P. Gawron, QI package for Mathematica, 2010, software freely available at http://zksi.iitis.pl/wiki/projects:mathematica-qi.
[8] B. Juliá-Díaz, F. Tabakin, QCWAVE, a mathematica quantum computer simulation update, http://www.pitt.edu/~tabakin/QW/.
[9] E. Gutiérrez, S. Romero, M.A. Trenas, E.L. Zapata, Quantum computer simulation using the cuda programming model, Comp. Phys. Comm. 181 (2) (2010) 283–300.
[10] I. Glendinning, B. Ömer, Parallelization of the QC-lib quantum computer simulator library, in: R. Wyrzykowski, et al. (Eds.), 5th International Conference on Parallel Processing and Applied Mathematics, in: LNCS, vol. 3019, 2004, pp. 461–468.
[11] K. De Raedt, K. Michielsen, H. De Raedt, B. Trieu, G. Arnold, M. Richter, Th. Lippert, H. Watanabe, N. Ito, Massively parallel quantum computer simulator, Comp. Phys. Comm. 176 (2) (2007) 121–136.
[12] F. Tabakin, B. Juliá-Díaz, QCMPI: A parallel environment for quantum computing, Comp. Phys. Comm. 180 (6) (2009) 948–964.
[13] M. Fürst, H. Weier, S. Nauerth, D.G. Marangon, Ch. Kurtsiefer, H. Weinfurter, High speed optical quantum random number generation, Optics Express 18 (12) (2010) 13029–13037.
[14] T. Heinosaari, M. Ziman, Guide to mathematical concepts of quantum theory, Acta Phys. Slovaca 58 (4) (2008) 487–674.
[15] K. Życzkowski, K.A. Penson, I. Nechita, B. Collins, Generating random density matrices, J. Math. Phys. 52 (2011) 062201.
[16] K. Życzkowski, H.-J. Sommers, Hilbert–Schmidt volume of the set of mixed quantum states, J. Phys. A: Math. Theor. 36 (39) (2003) 10115.
[17] K. Życzkowski, H.-J. Sommers, Average fidelity between random quantum states, Phys. Rev. A 71 (3) (2005) 032313.
[18] V. Al Osipov, H.-J. Sommers, K. Życzkowski, Random Bures mixed states and the distribution of their purity, J. Phys. A: Math. Theor. 43 (2010) 055302.
[19] ID Quantique SA, http://www.idquantique.com/.
[20] Quantis support page, http://www.idquantique.com/support/quantis-trng.html.
[21] W. Bruzda, V. Cappellini, H.-J. Sommers, K. Życzkowski, Random quantum operations, Phys. Lett. A 373 (3) (2009) 320.
[22] J.A. Miszczak, Singular value decomposition and matrix reorderings in quantum information theory, Int. J. Mod. Phys. C (2011), in press, doi:10.1142/S0129183111016683.
[23] D. Markham, J.A. Miszczak, Z. Puchała, K. Życzkowski, Quantum state discrimination: A geometric approach, Phys. Rev. A 77 (2008) 042111.
[24] B. Ömer, Quantum programming in QCL, Master's thesis, Vienna University of Technology, 2000.
[25] F.L. Marquezino, R. Portugal, The QWalk simulator of quantum walks, Comp. Phys. Comm. 179 (5) (2008) 359–369.
[26] T. Symul, S.M. Assad, P.K. Lam, Real time demonstration of high bitrate quantum random number generation with coherent laser light, Appl. Phys. Lett. 98 (23) (2011) 231103.
[27] Q.R.N.G. Service, Project developed by PicoQuant GmbH and the Nano-Optics groups at the Department of Physics of Humboldt University, https://qrng.physik.hu-berlin.de/.
[28] Quantum Random Bit Generator Service, Project developed by Centre for Informatics and Computing, Ruder Bošković Institute, Zagreb, Croatia, http://random.irb.hr/.