Available online at www.sciencedirect.com
SCIENCE DIRECT®
Computer Physics Communications 173 (2005) 91–113
Computer Physics Communications
www.elsevier.com/locate/cpc

Simulation of n-qubit quantum systems. I. Quantum registers and quantum gates ☆

T. Radtke, S. Fritzsche *

Institut für Physik, Universität Kassel, Heinrich-Plett-Str. 40, D-34132 Kassel, Germany

Received 17 February 2005; received in revised form 29 June 2005; accepted 12 July 2005

Available online 29 August 2005

Abstract

During recent years, quantum computations and the study of n-qubit quantum systems have attracted a lot of interest, both in theory and experiment. Apart from the promise of performing quantum computations, however, these investigations also revealed a great deal of difficulties which still need to be solved in practice. In quantum computing, unitary and non-unitary quantum operations act on a given set of qubits to form (entangled) states, in which the information is encoded by the overall system often referred to as quantum registers.

To facilitate the simulation of such n-qubit quantum systems, we present the FEYNMAN program to provide all necessary tools in order to define and to deal with quantum registers and quantum operations. Although the present version of the program is restricted to *unitary* transformations, it equally supports—whenever possible—the representation of the quantum registers both, in terms of their state vectors and density matrices. In addition to the composition of two or more quantum registers, moreover, the program also supports their *decomposition* into various parts by applying the partial trace operation and the concept of the reduced density matrix. Using an interactive design within the framework of MAPLE, therefore, we expect the FEYNMAN program to be helpful not only for teaching the basic elements of quantum computing but also for studying their physical realization in the future.

Program summary

Title of program: FEYNMAN
Catalogue number: ADWE
Program summary URL: http://cpc.cs.qub.ac.uk/summaries/ADWE
Program obtainable from: CPC Program Library, Queen's University of Belfast, N. Ireland
Licensing provisions: None

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

* Corresponding author.
E-mail addresses: t.radtke@physik.uni-kassel.de (T. Radtke), s.fritzsche@physik.uni-kassel.de (S. Fritzsche).

0010-4655/$ – see front matter
doi:10.1016/j.cpc.2005.07.006

Computers for which the program is designed: All computers with a license of the computer algebra system MAPLE [Maple is a registered trademark of Waterlo Maple Inc.]
Operating systems or monitors under which the program has been tested: Linux, MS Windows XP
Programming language used: MAPLE 9.5 (but should be compatible with 9.0 and 8.0, too)
Memory and time required to execute with typical data: Storage and time requirements critically depend on the number of qubits, n, in the quantum registers due to the exponential increase of the associated Hilbert space. In particular, complex algebraic operations may require large amounts of memory even for small qubit numbers. However, most of the standard commands (see Section 4 for simple examples) react promptly for up to five qubits on a normal single-processor machine ($\geqslant$ 1 GHz with 512 MB memory) and use less than 10 MB memory.
No. of lines in distributed program, including test data, etc.: 8864
No. of bytes in distributed program, including test data, etc.: 493 182
Distribution format: tar.gz
Nature of the physical problem: During the last decade, quantum computing has been found to provide a revolutionary new form of computation. The algorithms by Shor [P.W. Shor, SIAM J. Sci. Statist. Comput. 26 (1997) 1484] and Grover [L.K. Grover, Phys. Rev. Lett. 79 (1997) 325. [2]], for example, gave a first impression how one could solve problems in the future, that are intractable otherwise with all classical computers. Broadly speaking, quantum computing applies quantum logic gates (unitary transformations) on a given set of qubits, often referred to a quantum registers. Although, the theoretical foundation of quantum computing is now well understood, there are still many practical difficulties to be overcome for which (classical) simulations on n-qubit systems may help understand how quantum algorithms work in detail and what kind of physical systems and environments are most suitable for their realization.
Method of solution: Using the computer algebra system MAPLE, a set of procedures has been developed to define and to deal with n-qubit quantum registers and quantum logic gates. It provides a hierarchy of commands which can be applied interactively and which is flexible enough to incorporate non-unitary quantum operations and quantum error corrections models in the future.
Restrictions on the complexity of the problem: The present version of the program facilitates the set-up and manipulation of quantum registers by a large number of (predefined) quantum logic gates. In contrast to such *idealized* unitary transformations, however, less attention has been paid so far to non-unitary quantum operations or to the modeling of decoherence phenomena, although various suitable data structures are already designed and implemented in the code. A further restriction concerns the number of qubits, n, due to the exponentially growing time and memory requirements. Up to now, most of the complex commands are restricted to quantum registers with about 6 to 8 qubits, if use has to be made of a standard single-processor machine.
Unusual features of the program: The FEYNMAN program has been designed for interactive simulations on n-qubit quantum registers with no other restriction than given by the size and time resources of the computer. Apart from the standard quantum gates, as discussed in the literature [M.A. Nielsen, I.L. Chuang, Quantum Computation and Quantum Information, Cambridge University Press, Cambridge, 2000], it provides all the necessary tools to generalize these gates for n-qubits (in any given order of the individual qubits). Both common representations of the quantum registers in terms of their state vectors and/or density matrices are equally supported by the program whenever possible. In addition, the program also facilitates the composition of two or more quantum registers into a combined one as well as their decomposition into subsystems by using the partial trace and the use of the reduced density matrix for the individual parts.

PACS: 03.67.-a; 03.67.Lx

Keywords: Kronecker product; Quantum computation; Quantum logic gate; Quantum register; Qubit; (Reduced) density matrix; Unitary transformation

1. Introduction

Since Shor's algorithm [1] for factorizing large numbers, the theory of quantum computation and quantum information has attracted a lot of recent interest. This algorithm, in particular, has demonstrated that quantum computers may perform certain useful tasks (much) more *efficient* than their classical counterparts. Despite of the great promises of performing quantum computations, however, there are still many practical difficulties to be resolved before quantum computers might become available in the future. Hereby, many of the difficulties are

related closely to the (wanted and unwanted) interactions in n-qubit quantum systems as well as to the coupling of such systems with their environment.

Therefore, although the basic concepts of quantum computing are now well understood, their realization is likely not possible unless the (dynamical) behavior of n-qubit quantum systems, so-called quantum registers, can be simulated and analyzed in detail. For any successful implementation of quantum algorithms, of course, one has to provide and control the mechanisms for carrying out a sequence of computational steps, where each step k consists of a unitary transformation $U = \mathrm{e}^{-\frac{\mathrm{i}}{\hbar} H_k t_k}$ (as defined by some system Hamiltonian H_k and acting for a time t_k). To simulate such an *idealized* sequence of computational steps, a number of programs have been developed in recent years that all provide the underlying (linear-algebra) operations in a form as appropriate for general n-qubit quantum registers, although sometimes only for those registers with a fixed number of qubits.

A list of such (quantum computer simulation) programs can be found, for instance, in Refs. [3,4]. Often, however, these programs were designed for just a particular task, such as the demonstration of the superposition concept or the evaluation of (specific) quantum circuits and, hence, cannot be used for other applications. So far, most programs have been restricted to the *unitary* transformation of quantum registers being in a pure state, without the possibility to take into account 'mixtures' of quantum states (as to be described by density matrices) or the coupling of the quantum registers with their environment (quantum operations).

To facilitate the simulation of such general n-qubit quantum systems, here we present the FEYNMAN program that provides all necessary tools in order to deal with quantum registers and quantum operations. In contrast to most of the traditional programs from above [3,4], we followed an approach similar to [5], and designed and implemented the FEYNMAN code within the framework of MAPLE in order to take advantage of the (various) symbolic and numerical features of modern computer algebra. In a first version of our program, we provide a set of procedures to define quantum registers of variable size and to manipulate them by (time-independent) operators. Since a large number of such quantum operators have been predefined in the code, we expect the FEYNMAN program to be useful for quite different applications, both in education and research work.

In the next section, we first start with a brief account on the basic notations and concepts in the theory of quantum computations in order to facilitate the later use of the program. In Section 3, we then describe the program structure and how it is distributed. This includes a list of all user-accessible commands, together with short descriptions. Details about the parameters of each procedure, their optional arguments, etc. is provided by an additional user manual and is appended to the code. Section 4 illustrates the use of the FEYNMAN program by means of a few simple examples which may help display several central features and advantages of the code. Finally, a short outlook onto the current and possible future extensions of the program is given in Section 5.

2. Theoretical background

As the FEYNMAN program has been designed independent of any particular *physical realization* of an n-qubit quantum system, we may restrict our discussion of the theory to those notations and formulas as implemented in the program. Apart from the notion of the quantum register as the basic 'storage' to describe the behavior of general n-qubit (quantum) systems, emphasis is placed on the action of the various quantum gates and how they are 'distributed' over n qubits in order to transform the state of the system properly. Both representations of an n-qubit quantum system in terms of its state vector or density matrix are briefly explained below and are equally supported (whenever possible) by the program. A more detailed introduction into the theory of quantum computations and quantum information can be found, for instance, in the lecture notes by Preskill [6] and the textbook by Nielsen and Chuang [7].

2.1. Quantum bits and registers

2.1.1. Qubits and computational basis

In introducing the basic elements of quantum information theory, one often starts with the quantum bit (*qubit*) as the elementary 'unit' of information in dealing with quantum computations and the construction a quantum

computers. In contrast to the classical bit, which can take just one of two distinct values, say '0' and '1' or 'true' and 'false' (from the Boolean logic), respectively, the qubit is thought as a formal designation of a quantum system whose state vector is completely described by the superposition of two orthonormal eigenstates. Often, these eigenstates are labeled as $|0\rangle$ and $|1\rangle$ analogous to the binary basis used in classical digital computations. In practise, however, a qubit can be represented by any quantum mechanical two-state system including, for example, spin-1/2 particles with their two possible spin states $|\uparrow\rangle$ and $|\downarrow\rangle$, trapped ions, or by the charge state of quantum dots.

Independent of the physical realization of a particular qubit, there is an abstract (mathematical) notation

$$|\psi\rangle = a|0\rangle + b|1\rangle \equiv a\begin{pmatrix}1\\0\end{pmatrix} + b\begin{pmatrix}0\\1\end{pmatrix} = \begin{pmatrix}a\\b\end{pmatrix}, \tag{1}$$

in which the state $|\psi\rangle$ of the qubit is described as the linear superposition of the two states $|0\rangle$ and $|1\rangle$ from above. As usual, these states are supposed to be orthonormal and to form a basis of a two-dimensional Hilbert space $\mathcal{H}$. In this basis, the coefficients (amplitudes) a and $b \in \mathbb{C}$ are often called a *representation* of the qubit, while the squared absolute values $|a|^2$ and $|b|^2$ are known to provide the probability of finding the system in one or the other eigenstate in case of a measurement. That is, the coefficients a and b are supposed to fulfill the normalization $|a|^2 + |b|^2 = 1$. In the FEYNMAN program, the auxiliary procedure qbit() is used [cf. Section 3.2 below] in order to represent (the state of) a qubit by means of the complex coefficients a and b within the basis $\{|0\rangle, |1\rangle\}$, often denoted as the *computational basis*. Besides the computational basis, of course, any other orthonormal basis, such as the Hadamard basis $|+\rangle \equiv \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$ and $|-\rangle \equiv \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$, could be utilized equally well for all computations. However, unless stated otherwise we will always refer to the computational basis as the standard basis within the FEYNMAN program. The representation of a qubit in any other (orthonormal) basis is then obtained by performing the proper unitary transformation on the coefficients a and b.

Similar to classical computations, a single (qu-)bit is of little help in carrying out quantum computations. The generalization of a single qubit to n distinguishable and interacting qubits then leads us to the notion of a *quantum register* for which the computational basis is again the most natural choice in order to follow the (dynamical) behavior of n-qubit quantum systems. A rather large number of qubits is required not only for the implementation of useful quantum algorithms, but also for many quantum error-correcting codes (QECC) as necessary to protect n-qubit systems against the loss of information due to decoherence phenomena.

2.1.2. Quantum registers

The quantum-mechanical state of two or more (distinguishable) qubits can be described most easily if these qubits are arranged in some particular order and treated altogether in terms of a single quantum register. For an n-qubit quantum register, i.e. the collection of n individual qubits, then the state $|\Psi\rangle$ of the overall system is described by a vector in the 2^n-dimensional product space $\mathcal{H} = \mathcal{H}_1 \otimes \mathcal{H}_2 \otimes \cdots \otimes \mathcal{H}_n$ of the Hilbert spaces $\mathcal{H}_i$ $(i = 1, \ldots, n)$, associated with the individual qubits. Again, the 2^n basis states in $\mathcal{H}$ can be denoted analogously to the 2^n binary states of a classical n-bit register, $|00..00\rangle, |00..01\rangle, \ldots, |11..11\rangle$, but have to be formed as the (tensor) products of the corresponding computational basis states $\{|0\rangle_i, |1\rangle_i, i = 1, \ldots, n\}$ of the n qubits. In general, therefore, the state $|\Psi\rangle$ of the quantum register

$$|\Psi\rangle = \sum_{k=0}^{2^n-1} c_k |k\rangle_{\text{decimal}} = c_0 \begin{pmatrix}1\\0\\0\\\vdots\\0\end{pmatrix} + c_1 \begin{pmatrix}0\\1\\0\\\vdots\\0\end{pmatrix} + \cdots + c_{2^n-1} \begin{pmatrix}0\\0\\0\\\vdots\\1\end{pmatrix} \tag{2}$$

is written as a (normalized) superposition of the 2^n basis states. Note the rapid increase of the number of basis states (2^n) with the number of qubits, n, in the quantum register. Moreover, a seen from Eq. (2) for an equal superposition of all the 2^n basis states (i.e. if $c_k \equiv 1/\sqrt{2^n}$ for all k), a quantum computer would be able to process some given computation for all the 2^n values of a corresponding classical register simultaneously. It is this phenomenon,

sometimes known as *quantum parallelism*, which might provide quantum computers with the (computational) power to solve problems in the future, which are intractable otherwise for any classical computer.

For n *interacting* qubits, the superposition of the 2^n basis states in Eq. (2) generally results in a 'correlation' between the qubits, in which their individual (spin) states are no longer well-defined independently from the states of the other qubits. Such a correlation always occurs, if the state $|\Psi\rangle$ of the quantum register cannot be represented as a tensor product $|\Psi\rangle = |\psi_1\rangle \otimes |\psi_2\rangle \otimes \cdots \otimes |\psi_n\rangle$ of any particular states $|\psi_i\rangle$ $(i = 1, \ldots, n)$ of the individual qubits, a situation which often applies even if the qubits are well separated in space. In quantum mechanics, this (non-local) correlation phenomenon is known as *entanglement* and such a correlated state $|\Psi\rangle$ is said to be *inseparable*. As an example for two *entangled* qubits refer, for instance, to the Einstein–Podolsky–Rosen (EPR) states $(|00\rangle \pm |11\rangle)/\sqrt{2}$ which are often used in order to verify and explain the non-locality of quantum mechanics [8]. In fact, the possible entanglement in an n-qubit quantum register is considered to be one of the key ingredients of quantum computation as it is a *necessary* condition in order to achieve an exponential speed-up, when compared with classical computations [9]. During the past few years, therefore, the characterization and quantification of the entanglement (in terms of various useful *measures*) has been investigated intensively for different multi-qubit systems in the literature [10,11].

So far, we have always assumed in our discussion above that the state of the system is known completely, even if it is in an entangled state. Such a system, which is described by means of a state vector $|\Psi\rangle$ from Eq. (2) or a so-called 'ket'-vector, is said to be in a *pure state*. More general, however, one often has only incomplete knowledge about the state of the system, as it occurs in a statistical mixture $|\Phi\rangle = \sum_i p_i |\Psi_i\rangle$ of several (pure) states $|\Psi_i\rangle$ with some given probabilities $p_i \geqslant 0$ and $\sum_i p_i = 1$. To describe a quantum register in such a mixture of states, then the concept of the density operator needs to be applied in order to obtain a proper quantum-mechanical description [12]. In the computational basis, the density operator of the system is defined as $\rho = \sum_i p_i |\Psi_i\rangle\langle\Psi_i|$, where $\langle\Psi_i| = |\Psi_i\rangle^\dagger = (|\Psi_i\rangle^*)^{\mathrm{T}}$ denotes an element (a so-called 'bra'-vector) of the dual space. The density operator of a quantum system has a $2^n \times 2^n$ matrix representation which is self-adjoint $[\rho^\dagger = (\rho^*)^{\mathrm{T}} = \rho]$, positive semi-definite (non-negative eigenvalues), and which fulfills the trace condition $\mathrm{Tr}\,\rho = 1$. This density matrix, in addition, describes a pure state $\rho = |\Psi\rangle\langle\Psi|$, if and only if $\mathrm{Tr}(\rho^2) = 1$.

As outlined further in Section 2.2.3, the concept of the density operator is required in order to describe decoherence effects in *open* quantum systems which are coupled to some environment [16]. Although a description in terms of the density matrix is more general compared to the concept of pure states, it leads to a rapidly growing complexity and to quite sizeable storage requirements if more than only a few qubits are involved in the system. In the FEYNMAN program, both descriptions of a quantum system (register) in terms of its state vector or the density matrix are equally supported (whenever possible) by the auxiliary procedure qregister(), cf. Section 3.2. In fact, this procedure is the generalization of the qbit() command from above to the case of n correlated qubits, where the number of qubits is restricted only by local computer resources. As discussed below, this procedure provides the central (data) structure to describe the behavior of any pure or mixed n-qubit quantum register within the FEYNMAN program.

In describing an n-qubit quantum system, it is necessary not only to combine the various qubits and to describe them in terms of a (single) quantum register but also to split a multi-partite system into subsystems of different size. Such a splitting is useful, for example, if a quantum system is coupled to an environment or if a measurement needs to be performed on just a few of the qubits. To deal only with such a part of the system, there is the concept of the *reduced* density operator and the partial trace operation. For example, we consider a general composite (and probably entangled) system AB in the product Hilbert space $\mathcal{H}_{AB}$. The density matrix of the composite system is defined as

$$\rho^{AB} = \sum_{ik}^{N_A} \sum_{jl}^{N_B} \rho_{ij,kl} |i_A\rangle |j_B\rangle \langle k_A| \langle l_B|, \tag{3}$$

where $\{|i_A\rangle\}$ and $\{|j_B\rangle\}$ denote orthonormal bases for the subspaces $\mathcal{H}_A$ and $\mathcal{H}_B$ with dimension N_A and N_B, respectively.

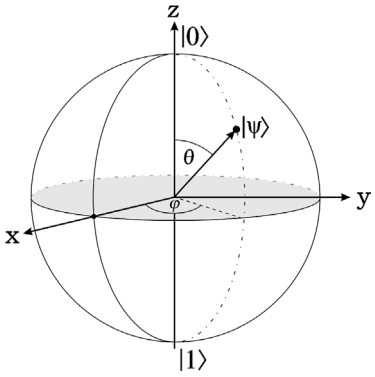

Fig. 1. Bloch sphere visualization of a (pure) one-qubit state.

The reduced density operator for the subsystem A is then defined as

$$\rho^A = \mathrm{Tr}_B\left(\rho^{AB}\right), \tag{4}$$

where $\mathrm{Tr}_B()$ is called the partial trace (operation) over the subsystem B. Using the completeness relation $\sum_m^{N_B} |m_B\rangle\langle m_B| = I$, the partial trace is given by

$$\mathrm{Tr}_B\left(\rho^{AB}\right) = \sum_{ik}^{N_A}\left(\sum_m^{N_B} \rho_{im,km}\right)|i_A\rangle\langle k_A|. \tag{5}$$

Since the trace over the space of the subsystem B is independent of the particular choice of the basis states $\{|j_B\rangle\}$, the partial trace is unique and appropriate to describe all the observables of the remaining system A. In the FEYNMAN program, the partial trace operation can be applied to any n-qubit quantum register in order to trace out one or several qubits by using the procedure Feynman_trace().

2.1.3. The Bloch sphere representation for single qubits

To visualize the state of a single qubit, there exists a common representation known as the Bloch sphere. In this representation, a (pure) one-qubit state is written as

$$|\psi\rangle = \cos\left(\frac{\theta}{2}\right)|0\rangle + \mathrm{e}^{\mathrm{i}\varphi}\sin\left(\frac{\theta}{2}\right)|1\rangle, \tag{6}$$

ignoring the overall phase which is physically irrelevant. Using the decomposition (6) of the state vector, the angles θ and φ are then taken as the 'polar' angles to describe the orientation of a real vector on the unit sphere, the so-called Bloch vector of the qubit. For some arbitrary single qubit pure state $|\psi\rangle$ the Bloch sphere representation is illustrated in Fig. 1.

Beside of pure states, the Bloch sphere representation can be used also for single qubits in a mixed state if their density operator ρ is re-written in terms of the Pauli spin matrices σ_i as $\rho(\vec{v}) = \frac{I+\vec{v}\cdot\vec{\sigma}}{2}$ with $\vec{\sigma} = (\sigma_x, \sigma_y, \sigma_z)$, and if the real vector $\vec{v}$ with $0 \leqslant |\vec{v}| \leqslant 1$ is now displayed in polar coordinates. For $|\vec{v}| = 1$, in particular, we always have $\mathrm{Tr}(\rho^2) = 1$ and thus a pure state (6) with the corresponding polar angles θ and φ. Using the Bloch-sphere representation, the action of any one-qubit (unitary) gate operation can be interpreted as a rotation of the Bloch vector. In the FEYNMAN program, a Bloch sphere representation of a single qubit (given possibly as part of an n-qubit quantum register) is obtained with the help of the Feynman_plot_Bloch_vector() procedure. Unfortunately, there is no straightforward generalization of this convenient representation known for multi-qubit systems.

2.2. Quantum gates and operations

Apart from keeping information and data in quantum registers, it should be possible also to manipulate these data in a controlled way. Analogous to classical computers, this can be achieved formally by applying a sequence of logical operations onto the state of the quantum systems, the so-called (quantum) gates or, more generally, the quantum operations. The standard gates are constructed in order to obtain a certain 'rotation' of the state vectors (density operators). In practise, the action of a quantum gate is often carried out by means of external fields, lasers, or by several other techniques depending on the particular experimental scheme used to realize the quantum register. Note that, beside of the external perturbation, here the natural evolution of the system due to its Hamiltonian needs to be taken into account. In the present version of the FEYNMAN program, however, we are not concerned with the physical realization of quantum registers and, hence, may ignore any explicit time evolution of the system. Instead, emphasis is placed on the action of the various (pre-defined) logical gates onto the states of general n-qubit quantum registers.

2.2.1. Unitary operations and quantum gates

Bennett [13] showed that any classical computation can be implemented also in a (logically and thermodynamically) reversible form. In fact, such a reversible implementation is possible even by using various rather small sets of logical operators, known as *universal* sets. Although, for the sake of simplicity, classical digital computers are actually not restricted to reversible operations, Bennett's concept has become important for the field of quantum computations and quantum information since, for any closed quantum system, the time evolution operator $U_{t_2,t_1} = \mathrm{e}^{-\frac{\mathrm{i}}{\hbar}H(t_2-t_1)}$ is unitary and, hence, represents a entirely *reversible* operation that maps a quantum state at time t_1 to the state at some other time t_2. For some pure state $|\psi\rangle$, therefore, we always have

$$\big|\psi(t_2)\big\rangle = U_{t_2,t_1}\big|\psi(t_1)\big\rangle \tag{7}$$

and, similarly,

$$\rho(t_2) = U_{t_2,t_1}\left(\sum_i p_i\big|\psi_i(t_1)\big\rangle\big\langle\psi_i(t_1)\big|\right)U^\dagger_{t_2,t_1} = U_{t_2,t_1}\rho(t_1)U^\dagger_{t_2,t_1} \tag{8}$$

if the state of the system is given by a statistical mixture $\{p_i, |\psi_i\rangle\}$. In a quantum computer, consequently, it is necessary to implement all the computational steps by means of reversible and unitary operations, taking into account also the modified Hamiltonian of the underlying (physical) system. Formally, Bennett's result ensures that a quantum computer is—at least in principle—capable of performing also every classical algorithm one can think of, although typically without any gain in speed compared to classical computers.

In general, any algorithm on an n-qubit quantum register could be performed within a single step by acting (in a highly non-trivial way) with a unitary operation on all the n qubits of the register simultaneously. Similar to classical computations, however, it is more convenient in practise if such operations are decomposed into a sequence of one- and two-qubit operations (gates), taken from a small set of *universal gates* which is appropriate for a particular physical realization of the system. In the context of the FEYNMAN program, here we need not discuss which universal set is most suitable for a given experimental setup. In a formal treatment of quantum algorithms and transformations, instead, one often uses the two-qubit controlled-not (CNOT) gate together with any one-qubit gate (constructed from the Pauli sigma matrices) as the basic elements into which then all the quantum gates are decomposed [14]. More generally, it has been shown also that *any* (non-separable) two-qubit gate is *universal* for quantum computations if combined with some one-qubit gate [15]. The need of a non-separable two-qubit gate again highlights the key role of entanglement for performing useful quantum computations.

In order to apply a (unitary) gate operation to just one or a few individual qubits of a given quantum register, we need to create operators which act only on the subspaces associated to these qubits. First, therefore, let us briefly recall here the definition of the tensor product for matrix operators. Suppose A and B are two linear operators in

the subspaces $\mathcal{H}_1$ and $\mathcal{H}_2$, respectively. Then $A \otimes B$ is used to denote a linear operator acting on the product space $\mathcal{H}_1 \otimes \mathcal{H}_2$ and is defined by

$$(A \otimes B)\big(|\psi_1\rangle \otimes |\psi_2\rangle\big) = A|\psi_1\rangle \otimes B|\psi_2\rangle. \tag{9}$$

Using a matrix representation for the operators A and B, the tensor (or Kronecker) product of an $m \times n$ matrix A and a $p \times q$ matrix B is then given by the $mp \times nq$ matrix

$$A \otimes B = \begin{pmatrix} a_{11}B & \cdots & a_{1n}B \\ \vdots & \ddots & \vdots \\ a_{m1}B & \cdots & a_{mn}B \end{pmatrix}, \tag{10}$$

where a_{ij} are the matrix elements of A. For instance, a single-qubit *not* or X-gate

$$X \equiv \sigma_x = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \tag{11}$$

that acts onto the second qubit within a three-qubit quantum register can be expressed as the tensor product

$$U = I \otimes X \otimes I, \tag{12}$$

if the two other qubits are left unchanged and if I denotes the 2×2 identity operator acting on a single qubit. In the FEYNMAN program, the matrix representations of many widely used quantum gates are pre-defined already in the main procedure Feynman_quantum_operator(); see Table 1, for instance, for the pre-defined one-qubit gates. More detailed information about all the pre-defined quantum gates in the FEYNMAN program, including their matrix representation as well as circuit symbols, can be found in the manual `Feynman-commands.pdf` which is provided together with the code.

2.2.2. *'Distributed' quantum gates*

In the previous section, we introduced the use of single and multi-qubit quantum gates, and how they can be combined with the single-qubit identity I in order to 'act' on the space associated with some n-qubit quantum

Table 1
Single-qubit quantum operators (gates) as accessible in the program by the command Feynman_quantum_operator()[1]

Argument option	Output: returns the matrix representation of
`("I")`	identity operator
`("not")` or `("sigma_x")` or `("X")`	$X \equiv \sigma_x$
`("sigma_y")` or `("Y")`	$Y \equiv \sigma_y$
`("sigma_z")` or `("Z")`	$Z \equiv \sigma_z$
`("sigma[+]")`	$\sigma_+ = \sigma_x + \mathrm{i}\sigma_y$ (not unitary!)
`("sigma[-]")`	$\sigma_- = \sigma_x - \mathrm{i}\sigma_y$ (not unitary!)
`("phase",`ϕ`)`	relative phase shift by ϕ
`("A",`ϕ`)`	alternative definition of the phase gate
`("Hadamard")` or `("H")`	H
`("Euler",[`$\alpha,\beta,\gamma,\delta$`])`	$U = \mathrm{e}^{\mathrm{i}\alpha} R_z(\beta) R_y(\gamma) R_z(\delta)$
`("Rx",`θ`)`	$R_x(\theta)$ rotation
`("Ry",`θ`)`	$R_y(\theta)$ rotation
`("Rz",`θ`)`	$R_z(\theta)$ rotation
`("S")`	phase gate with $\phi = \pi/2$
`("T")`	$\pi/8$ gate T
`("not^1/2")`	$\sqrt{NOT}$

[1] A complete list of all available single and multi-qubit quantum gates together with their definition are provided by the manual `Feynman-commands.pdf` which is distributed with the code.

Table 2
Distributed two-qubit quantum operators as accessible by the command Feynman_quantum_operator()

Argument option	Output: returns the matrix representation of
`(n,"cn",[`m_1`, `m_2`])` or `(n,"cnot",[`m_1`, `m_2`])`	*controlled-not* operation on the qubits m_1 and m_2 of an n-qubit quantum system
`(n,"controlled-phase",[`m_1`, `m_2`])` or `(n,"cs",[`m_1`, `m_2`])`	*controlled-phase* or *cs* operation on the qubits m_1 and m_2 of an n-qubit quantum system
`(n,"controlled-z",[`m_1`, `m_2`])` or `(n,"cz",[`m_1`, `m_2`])`	*controlled-z* or *cz* operation on the qubits m_1 and m_2
`(n,"swap",[`m_1`, `m_2`])`	*swap* operation on the qubits m_1 and m_2

register. To this aim, the use of the tensor product [cf. Eqs. (9) and (12)] is obvious as long as the affected qubits are neighbors with respect to the numbering of qubits in the quantum register or, respectively, the computational basis. In contrast, if we consider some k-qubit quantum gate to act onto $k \leqslant n$ not adjacent qubits in an n-qubit register, the 2^k matrix elements of the k-qubit quantum gate have to be 'distributed' properly over the whole $2^n \times 2^n$ matrix which acts on the complete n-qubit quantum register. To this end, here we introduce the notion of a *distributed* gate if a (k-qubit) gate is applied to $k \leqslant n$ selected qubits, given in a sequence $m_1, m_2, \ldots, m_k$ and with $m_i \leqslant n$ for all selected indices i of the qubits.

For example, the controlled-not (CNOT) gate is known to invert the state of the target qubit in dependence of the state of control qubit. Its matrix representation is given by

$$\mathrm{CNOT} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}, \tag{13}$$

if the first qubit refers to the control qubit and the second to the target qubit. For instance, if we wish to apply this CNOT gate to the first and third qubit in a three-qubit quantum register, the use of the tensor product (9) above is not directly possible unless the qubits are first be renumbered in the quantum register (and that the state vector or density matrix of the system is transformed accordingly). Although such a permutation of the qubits is certainly quite standard and is achieved simply by transforming both, the operator and the quantum register into a computational basis which refers to the new sequence of qubits, in the FEYNMAN program we provide a special syntax for the procedure Feynman_quantum_operator() that allows the user to 'distribute' any k-qubit operator over an n-qubit quantum gate just by specifying the sequence of the $k < n$ qubits to be concerned. Table 2 displays the predefined two-qubit gates in the FEYNMAN program and how they can be distributed in order to obtain a proper n-qubit unitary matrix.

2.2.3. Quantum operations

The quantum gates from above refer to an *idealized* quantum computer whose (quantum) registers evolve like any *closed* quantum system entirely unitary according to the known Hamiltonian of the system. This is quite in contrast, of course, to any real implementation where various unwanted interactions of the quantum register with its environment, such as a spontaneous emission of photons from the system or some other form of energy and phase dissipation, may occur. Frankly speaking, such a coupling to the environment leads to a non-unitary and hence irreversible evolution of the quantum computer (the principal system) as it is now only a subsystem of the overall system 'quantum computer + environment' [16]. Or, in other words, the entanglement with the environment finally results in a reduced density matrix of the principal system which is in a *mixed* state [cf. Section 4.3] and which is associated with a loss of information (known also as decoherence).

In order to allow a non-unitary evolution of quantum systems (registers), there exists the general formalism of quantum operations to which one sometimes refers also as superoperators or completely positive maps. Since, this formalism is not yet fully supported by the FEYNMAN program, here we will give only a brief account on

this concept and refer for all further details to the literature [7]. In the program, however, a first step towards the incorporation of quantum operations has been done by providing an appropriate data structure in Section 3.2. If we consider, for instance, a principal system S and the environment E, a 'noisy' interaction between S and E is described by a (quantum dynamical) map $\mathcal{E}$ that takes a system state ρ to another density matrix $\rho' = \mathcal{E}(\rho)$. In order to ensure that ρ' is a valid density operator, it can be shown that $\mathcal{E}$ acting on $\mathcal{H}_S$ must be *completely positive*, i.e. $\mathcal{E}(A)$ must be positive for any positive (density) operator A and, moreover, if we extend $\mathcal{H}_S$ by a second Hilbert space $\mathcal{H}_E$ of arbitrary dimension the combined operation $\mathcal{E} \otimes \mathcal{I}$ ($\mathcal{I}$ being the identity operation on $\mathcal{H}_E$) acting on product space $\mathcal{H}_S \otimes \mathcal{H}_E$ still maps any positive operator of the composite system to a positive operator. In the (so-called) operator-sum representation, such a map is described by [7,17]

$$\rho' = \mathcal{E}(\rho) = \sum_i E_i \rho E_i^\dagger, \tag{14}$$

where E_i denotes an operation element (or Kraus operator) with $\sum_i E_i^\dagger E_i \leqslant I$ and where the equal sign applies for all trace-preserving quantum operations. From the definition (14) of the quantum operations it is clear, that this includes as a special case also the *unitary* evolution of the quantum system from Section 2.2.1, if the quantum operation is given by a single unitary operation element. With the further support of quantum operations by the FEYNMAN program, therefore, it will be possible to implement and to follow-up also various decoherence models for n-qubit quantum registers as discussed recently in the literature [18].

3. Program organization

3.1. Overview

The FEYNMAN program has been designed to support the simulation of n-qubit quantum systems (quantum registers). Apart from the definition and initialization of quantum registers, this requires to transform their state either by unitary or non-unitary operations until the result of the computations can be *read off* (i.e. measured) from the register. For any realization of quantum registers, this makes it necessary within the simulation to include the coupling of the system to its environment, both for wanted and unwanted interactions. However, not all of these practical 'requirements' can be realized in a first implementation of a program. In the present version of the FEYNMAN program, our aim is to establish the basic data structures and to provide a simple access to the unitary transformation of n-qubit quantum registers with no further restriction on n other than given by the memory and time-limitations of the computer.

As mentioned before, the FEYNMAN program is in several respects different from other codes. Although our program is presently restricted to perform *unitary* transformations of quantum registers, it equally supports—whenever possible—the representation of their states either in terms of state vectors or density matrices. In composition of two or more quantum registers, addition to, the program also supports their *decomposition* into various parts by applying the partial trace operation and the concept of the reduced density matrix. A further advantage is that the FEYNMAN program provides an interactive tool for which the knowledge of a few (main) procedures is enough to carry out most of the computations. When compared to a purely numerical implementation using, for instance, C++ or JAVA, the FEYNMAN program enables the user to perform the computation either in a symbolic or numerical form, without that much extra code has to be developed and tested. Finally, since MAPLE by itself offers a large set of built-in mathematical functions, this may help extend the program into various directions in the future, including topics such as non-unitary quantum operations, quantum measures, decoherence, error correction, and several others.

Following MAPLE's philosophy, the FEYNMAN program has been organized as a hierarchy of currently about 25 procedures at different level of complexity. Apart from several low-level subprocedures, which remain hidden to the user, the main body of the procedures can be used either for interactive work or simply as a language element

in order to build-up commands at some higher level of the hierarchy. As discussed below, these procedures are divided into two groups, the *auxiliary* procedures as the building blocks (and data structures) of the program as well as the *main* commands, which help operate on this structures. A list of all user-accessible procedures are displayed in Tables 3 and 4; as seen from these tables, use is made of rather long names in order to improve the readability and the maintenance of the program. Moreover, in order to distinguish the FEYNMAN procedures from MAPLE's internal ones, all names of the main commands start with the prefix Feynman_. A more detailed description of some selected commands, including the argument options and some additional information, can be found in Appendices A and B which follow the style of the help pages of MAPLE and *The Maple Handbook*

Table 3
Auxiliary procedures of the FEYNMAN program to represent the basic data structures. These procedures are utilized mainly for keeping related information together and to facilitate the data handling with and within the program

Procedure	Explanation
cbs()	To represent a computational basis state $\|k\rangle_{\rm dec}$ of an n-qubit quantum register in the decimal basis $\|0\rangle, \|1\rangle, \ldots, \|k\rangle, \ldots, \|2^n - 1\rangle$
qbit()	To represent an one-qubit state $\|\psi\rangle = a\|0\rangle + b\|1\rangle$ in terms of the two (complex) coefficients a and b in the computational basis $\|0\rangle \equiv \binom{1}{0}$ and $\|1\rangle \equiv \binom{0}{1}$, respectively
qregister()	To represent an n-qubit quantum register either in terms of its 2^n-dimensional state vector or the $2^n \times 2^n$ density matrix
qoperator()	To represent an n-qubit quantum logic gate or quantum operator (unitary transformation)
qoperation()	To represent an n-qubit quantum operation $\mathcal{E}(\rho) = \sum_i E_i \rho E_i^\dagger$ by means of its operation elements (Kraus operators) E_i

Table 4
Main procedures of the FEYNMAN program as available by the user for interactive work. For three selected commands (marked by an * below), a more detailed explanation is displayed in Appendix B[2]

Procedure	Explanation
Feynman_apply()	Applies a given qoperator() or qoperation() to the state vector or density matrix of an n-qubit quantum register
Feynman_norm()	Calculates the norm (or trace) of the state vector or density matrix of a quantum register
Feynman_normalize()	Normalizes the state vector or density matrix of a quantum register
Feynman_operator_function()*	Evaluates an operator function for a given matrix or qoperator() and returns its explicit matrix representation
Feynman_operator_type()*	Determines whether a given matrix or qoperator() has some particular property (hermitian, normal, etc.)
Feynman_plot_Bloch_vector()	Returns a 3D plot of the Bloch-sphere representation for a single qbit() or for a selected qubit within a given qregister()
Feynman_plot_probability()	Returns a 2D (or 3D) histogram plot of the (squared) amplitudes for a pure or mixed state as represented by the qregister()
Feynman_print()	Prints the state vector or the density matrix of a quantum register in Dirac notation (by using the standard computational basis)
Feynman_product()*	Carries out several types of product operations (inner, outer, Kronecker, Hadamard, etc.) for two or more given quantum operators and/or quantum registers, respectively
Feynman_qgate()	Carries out some pre-defined quantum gate on a sequence of qbit()s
Feynman_quantum_operator()	Evaluates the explicit matrix representation of various pre-defined and distributed one-, two-, three-, or n-qubit quantum operators
Feynman_set_qregister()	Returns a qregister() in some (pre-defined) state such as the computational basis states, the Bell states, or several others
Feynman_trace()	Calculates the reduced density operators of a qregister() (i.e. the partial trace) and the expectation values for given matrix operators

[2] A complete description of all user-accessible (exported) commands is provided by the manual `Feynman-commands.pdf` which is distributed together with the code.

by Redfern [19] from earlier years. A complete description of the program is given in the manual `Feynman-commands.pdf` and is distributed together with the program.

Beside the benefits in using MAPLE, there are a few drawbacks concerning the performance of the FEYNMAN program. Even if purely numerical computations are to be carried out, our procedures are slower by about 1–2 orders of magnitude when compared with most traditional languages. This limitation and the storage management of most CAS presently restricts the number of qubits to about 6–8 if complex computations are performed. To improve the performance of the program, either a 'translation' of the code into some compilable programming language or the concept of decision diagrams [20] could be used but are currently not supported by the program.

3.2. Auxiliary procedures

Any (automatic) transformation of an n-qubit quantum system due to quantum logic gates, circuits, or even quantum algorithms must provide the user with a fast and simple access to the underlying quantum-mechanical objects such as the quantum register or some (unitary) transformation. In fact, these registers and gates are the central 'building blocks' which allow one to simulate the behavior of general (n-qubit) systems in quantum computing. To facilitate the handling of these and similar (data) structures, a number of *auxiliary* procedures have been defined in the FEYNMAN program to deal with qubits, quantum registers, quantum operators, and several other structures as described in Section 2; cf. Table 3.

The use of such auxiliary procedures, which return their arguments *unevaluated*, has several merits. Apart from having related information together, it clearly simplifies the communication with and within the program. The procedure qregister(), for example, represents a general n-qubit quantum register in either a pure or mixed state—including the information about the number of qubits, an identifier, as well as the state vector or density matrix of the register—, without that the user needs to be concerned about of these details. Similarly, the qoperator() command is used to specify a particular quantum operator (unitary transformation) by means of a *keyword* or a matrix, if given explicitly. The use of keywords, together with the designation of the qubits on which the operator acts, will later simplify the definition of quantum circuits and algorithms. In practice, there is a large number of predefined gates available in the FEYNMAN program which are taken from an internal 'gate library' and distributed over any given number n of qubits by using the command Feynman_quantum_operator(), see below.

In addition to quantum registers, quantum operations, etc., there is the procedure cbs() to represent a computational basis state. This auxiliary procedure is typically used in the input of some main commands to refer to some particular basis states such as $|0\rangle$ or $|1011\rangle$ (given either in decimal or binary notation). Together with the command Feynman_set_qregister(), a call to the procedures cbs() helps set-up and initialize a quantum register in a well-defined state at the beginning of the simulation.

3.3. Main commands

Having defined some proper 'data structures' above, the *main* procedures are provided to set-up and manipulate these data according to the rules of quantum computing. Apart from the unitary transformation of some quantum register, these commands enable one to compose or to reduce quantum registers and operators, or to visualize (plot) the state of qubits and quantum registers. The command Feynman_plot_probability(), for example, returns a histogram of the (squared) amplitudes for the pure or mixed state of a quantum register as represented by qregister(). Table 4 shows a list of all the commands which are presently accessible by the user. For a few selected commands, in addition, a more detailed explanation can be found in Appendix B.

There are several ways to define and to 'initialize' the state of an n-qubit quantum register. Beside its explicit construction by means of the qubit() command, there is the Feynman_set_qregister() which helps to create quantum registers in various (pre-defined) states such as the computational basis states, the Bell states, or various others. Separable states of a quantum register, moreover, can be composed as Kronecker products of two or more quantum

registers by using the command Feynman_product(). Typically, however, it is better to start from either a computational basis state $|k\rangle_{\mathrm{dec}}$ or a randomized quantum register to which then a proper set of transformations is applied for initialization.

There are two procedures in the FEYNMAN program which provide (and perform) unitary transformations. While the command Feynman_qgate() is designed to carry out a (*keyword*-defined) quantum gate directly onto a given sequence of independent qubits(), the procedure Feynman_quantum_operator() returns the explicit matrix representation of these gates, distributed due to the designation of the target qubits. More easily, however, the same transformation is often obtained by means of Feynman_apply() with which one may act with a qoperation() or qoperator() directly onto the state vector or the density matrix of a given quantum register [qregister()].

Finally, to reduce the density operator of a given qregister(), the command Feynman_trace() can be invoked to calculate the *partial* trace over one or several qubits. The same procedure also allows to calculate the expectation value of a matrix operator with respect to a given quantum state. Further commands concerning the use of non-unitary operations or time-dependent transformations will be 'added' later in a forthcoming version of the program.

3.4. Distribution of the program

Following MAPLE's recent developments, the FEYNMAN program is provided as a library which can be loaded by the `with(Feynman)` command. This library contains all the procedures as listed in Tables 3 and 4, along with several subprocedures which remain *hidden* to the user. From the CPC library, the FEYNMAN program is distributed as a (compressed) `zip`-file `feynman.zip` from which the `Feynman` root directory can be obtained. This root directory contains the library files, the source code and a `Read.me` for the installation of the package as well as the program manual `Feynman-commands.pdf`. This latter document explains all the user relevant commands along with the output format, their argument options as well as various additional information which is of interest for the application of the procedures. The `Feynman` root directory also contains an example of a `.mapleinit` file which can be modified and incorporated into the user's home. Making use of such a `.mapleinit` file, the Feynman library should then be available like any other module of MAPLE.

4. Interactive work using the FEYNMAN procedures: Examples

To illustrate the interactive use of the FEYNMAN procedures, a few simple examples are displayed below as they might occur, for instance, in an introductory course on quantum computation. Apart from these test cases, of course, the same procedures can be utilized also for dealing with more complex tasks. Since, however, some of the procedures result in a rather large MAPLE output, a colon (instead of a semicolon) is used below at the end of several commands in order to restrict the printed output in this section.

4.1. Identity of simple quantum circuits

In a first example, let us access some of the pre-defined (one-qubit) quantum gates of the FEYNMAN program and verify, that they fulfill the two well-known relations $XYX = -Y$ and $XR_y(\theta)X = R_y(-\theta)$, respectively [cf. Ref. [7], exercise 4.7]. To this end, we just enter at MAPLE's prompt (having loaded the FEYNMAN module before)

```
> X := Feynman_quantum_operator("X");

                [0    1]
           X := [      ]
                [1    0]
```

```
> Y := Feynman_quantum_operator("Y");

            [0    -I]
       Y := [       ]
            [I    0 ]
```

and form the product XYX as usual by

```
> X.Y.X;

       [0     I]
       [       ]
       [-I    0]
```

which confirms this identity immediately. For the second relation, similarly, we may type

```
> Ry := theta -> Feynman_quantum_operator("Ry", theta);

        Ry := theta -> Feynman_quantum_operator("Ry", theta)

> X.Ry(theta).X,   Ry(-theta);

        [    theta          theta ]  [    theta          theta ]
        [cos(-----)     sin(-----)]  [cos(-----)     sin(-----)]
        [      2              2   ]  [      2              2   ]
        [                        ], [                        ]
        [     theta          theta ]  [     theta          theta ]
        [-sin(-----)    cos(-----)]  [-sin(-----)    cos(-----)]
        [       2              2   ]  [       2              2   ]
```

to obtain the expected result. For one-qubit quantum gates, in addition, it was proved that their 2×2 unitary matrices can be expressed always in terms of rotations around the three Cartesian axes, together with an overall phase factor $e^{i\phi}$ with $0 \leqslant \phi < 2\pi$ [cf. Ref. [7], exercise 4.4]. As another short example, therefore, let us confirm explicitly that the Hadamard gate, $H = \frac{1}{\sqrt{2}}\left(\begin{smallmatrix} 1 & 1 \\ 1 & -1 \end{smallmatrix}\right)$, is equivalent to a rotation around the y-axis by the angle $\pi/2$, followed by a rotation around the (former) x-axis by π, and multiplied by a phase factor which still needs to be determined. Within the FEYNMAN program, this angle is easily derived by first constructing the matrix $U = e^{i\phi} R_x(\pi) R_y(\pi/2)$:

```
> Ry := alpha -> Feynman_quantum_operator("Ry",alpha):

> Rx := beta -> Feynman_quantum_operator("Rx",beta):

> U := exp(I*phi) * (Rx(Pi).Ry(Pi/2));

            [                   1/2                         1/2]
            [-1/2 I exp(phi I) 2       -1/2 I exp(phi I) 2   ]
       U := [                                                 ]
            [                   1/2                        1/2 ]
            [-1/2 I exp(phi I) 2       1/2 I exp(phi I) 2    ]
```

and by solving the (matrix) equation $U = H$ with respect to the phase ϕ, i.e. for all the matrix elements simultaneously

```
> H := Feynman_quantum_operator("H"):

> solve({U[1,1]=H[1,1], U[1,2]=H[1,2], U[2,1]=H[2,1], U[2,2]=H[2,2]}, phi);

        {phi = 1/2*Pi}
```

Apparently, the angle $\phi = \pi/2$ fulfills all the four equations and can be used to make U equivalent to the Hadamard gate.

```
> subs(phi=Pi/2, U);

       [ 1/2      1/2 ]
       [2        2    ]
       [----     ---- ]
       [ 2        2   ]
       [                ]
       [ 1/2       1/2 ]
       [2         2    ]
       [----     ---- ]
       [ 2         2   ]
```

Using similar steps, of course, a large number of other identities can be shown (or found) which would require rather lengthy computations otherwise.

4.2. Probability of measuring a computational basis state

So far, we just dealt with one-qubit quantum gates, for which most of the matrix operations could be easily carried out by hand. In practise, however, the complexity of such operations increases rapidly as quantum registers with several qubits occur in some computation. For the sake of simplicity, let us consider here the case of two qubits A and B which, initially, are both supposed to be in the state $|\psi_A\rangle = |\psi_B\rangle = (|0\rangle + |1\rangle)/\sqrt{2}$, taken in the computational basis of their corresponding Hilbert spaces $\mathcal{H}_i$. Let us further consider the composite state of these qubits, i.e. the tensor product $|\psi\rangle = |\psi_A\rangle \otimes |\psi_B\rangle$, and the matrix

$$U = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & -1 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & -1 & -1 \\ 0 & 0 & 1 & -1 \end{pmatrix}$$

which acts on both qubits, $U|\psi\rangle = |\phi\rangle$, in order to form a (two-qubit) state. For this state $|\phi\rangle$, we may ask for the probability to 'find', i.e. to measure, the (computational) basis state $|2\rangle_{\text{dec}} = |10\rangle$. Mathematically, this requires of course to calculate one of the expressions

$$\text{Tr}\big(|\phi\rangle\langle\phi|P_{10}\big) = \langle\phi|P_{10}|\phi\rangle, \tag{15}$$

which are known to be equivalent for any *pure* state $|\phi\rangle$ of the system. Within the FEYNMAN program, there exist several ways to accomplish such computations owing to the (pre-)definition and the combination of various quantum registers (qregister) and quantum operators (qoperators) in the code. Starting from the single-qubit notation above, we may construct the two-qubit states $|\psi\rangle$ and $|\phi\rangle$ by typing

```
> psi[A] := qregister(id, qbit(id, 1/sqrt(2), 1/sqrt(2))):

> psi[B] := qregister(id, qbit(id, 1/sqrt(2), 1/sqrt(2))):
```

```
> psi := Feynman_product("Kronecker", psi[A], psi[B]);

                                [1/2]
                                [   ]
                                [1/2]
          psi := qregister(id, 2, [   ])
                                [1/2]
                                [   ]
                                [1/2]
```

and

```
> U := qoperator(1/sqrt(2)*Matrix([[1,-1,0,0], [1,1,0,0], [0,0,-1,-1],
       [0,0,1,-1]])):

> phi := Feynman_apply(U, psi);

                                [  0   ]
                                [      ]
                                [  1/2 ]
                                [ 2    ]
                                [ ---- ]
                                [  2   ]
          phi := qregister(id, 2, [      ])
                                [   1/2]
                                [  2   ]
                                [- ----]
                                [   2  ]
                                [      ]
                                [  0   ]
```

respectively. The projection operator P_{10}, moreover, is given by the 2-qubit density operator $P_{10} = |10\rangle\langle 10|$, or simply the outer product of the computational basis state

$$|10\rangle = |1\rangle \otimes |0\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix} \otimes \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix}$$

with itself.

```
> aux_vector := Vector([0,0,1,0]):
> P[10] := Feynman_product("outer", aux_vector, aux_vector);

                      [0    0    0    0]
                      [                ]
                      [0    0    0    0]
            P[10] := [                ]
                      [0    0    1    0]
                      [                ]
                      [0    0    0    0]
```

From P_{10}, the expectation value $\langle\phi|P_{10}|\phi\rangle = \langle\phi|10\rangle\langle 10|\phi\rangle$ respectively the probability for measuring the 2-qubit system in the basis state $|10\rangle$ is then calculated as the inner product of $|\phi\rangle$ and the auxiliary state $P_{10}|\phi\rangle$

```
> aux := Feynman_apply(qoperator(P[10]), phi):

> Feynman_product("inner", phi, aux);

        1/2
```

if all the necessary steps are carried out explicitly. In the FEYNMAN program, the same computation can be performed much shorter because a large number of frequently required quantum states and (projection) operators are pre-defined already in the code. Since we have $|\psi_A\rangle = |\psi_B\rangle = |+\rangle$ and, hence, $|\psi_A\rangle \otimes |\psi_B\rangle = |{+}{+}\rangle$, we could create the starting state $|\psi\rangle$ within a single line:

```
> psi := Feynman_set_qregister("++");

                                  [1/2]
                                  [   ]
                                  [1/2]
          psi := qregister("++", 2, [   ])
                                  [1/2]
                                  [   ]
                                  [1/2]
```

... and similarly for the projection operator P_{10}:

```
> P[10] := Feynman_quantum_operator("projector", cbs("10")):
```

The transformation $|\phi\rangle = U|\psi\rangle$ is then carried out as above, using the Feynman_apply() command.

Now the probability $\mathrm{Tr}(|\phi\rangle\langle\phi|P_{10})$ can be calculated by means of

```
> Feynman_trace(P[10], phi);

       1/2.
```

This second way in doing the computations 'proves' also that the relation (15) is fulfilled for the pure state $|\phi\rangle$ from above.

4.3. Partial trace of a Bell state

In our last example, finally, we demonstrate how the partial trace can be calculated for the density matrix of a composite system. Suppose we have the bipartite system AB and need to investigate the behavior of the subsystem A without that the system B is observed, i.e. if we wish to 'trace out' the dependence on B. For a *product* state $\rho^{AB} = \rho^A \otimes \rho^B$, the density matrix of the system A is obtained simply by taking the trace over ρ^B

$$\rho^A = \mathrm{Tr}_B\left(\rho^A \otimes \rho^B\right) = \rho^A \,\mathrm{Tr}\left(\rho^B\right),$$

which, however, does not apply for any *entangled* state. In the case of an entangled system, the reduced density operators of the subsystems cannot be read off so easily but requires one to carry out the partial trace operation (5) explicitly. If, for example, we consider the composite system AB in the Bell state $|\Phi^+\rangle = \frac{|00\rangle+|11\rangle}{\sqrt{2}}$

```
> Phi := Feynman_set_qregister("Bell","Phi+");

                                    [ 1/2]
                                    [2   ]
                                    [----]
                                    [ 2  ]
                                    [    ]
                                    [ 0  ]
          Phi := qregister("Phi+", 2, [    ])
                                    [ 0  ]
                                    [    ]
                                    [ 1/2]
                                    [2   ]
                                    [----]
                                    [ 2  ]
```

we may calculate the density matrix ρ^A of the first qubit by tracing out the second qubit:

```
> rho[A] := Feynman_trace(Phi,[2]);

                                      [1/2     0 ]
             rho[A] := qregister(id, 1, [        ])
                                      [ 0     1/2]
```

From the command Feynman_trace(), this matrix ρ^A is then obtained as the third argument of the auxiliary procedure qregister(), in line with the (internal) use of quantum registers to represent either a pure or mixed state of an n-qubit system. Taking the trace over the squared reduced matrix, $\mathrm{Tr}(\rho^A)^2 = \frac{1}{2}$, we easily see that the subsystem A is in a *mixed* state, and that the same applies also for the subsystem B owing to the symmetry of the Bell state $|\Phi^+\rangle$. Therefore, although the composite system was originally in a pure quantum state, the two subsystems A and B by themselves are statistical mixtures of states, a result which is well known from the literature [7] but is worth here to show again explicitly. The same fact applies for any *open* quantum system, if a principal system A is *entangled* with its environment B (see Section 2.2.3).

5. Summary and outlook

With the FEYNMAN program, an interactive and flexible tool has been developed for the analysis and simulation of n-qubit quantum systems. In a first version of this program, our aim was to design the basic (data) structures for quantum computations, such as quantum registers and quantum gates, and to provide many of the elementary operations which are needed in order to work with these data structures. Using the computer algebra system MAPLE as the computational framework, moreover, we are able to support both, symbolic and numerical computations as well as various hybrid forms, in contrast to many other (commercially or free-accessible) *simulators* which are available today for doing quantum computations.

There are several extensions of the FEYNMAN program which are desirable for future applications. Apart from quantum registers and operators as defined above, the implementation and the support of quantum operations and quantum circuits (as additional data structure) might help in the analysis and optimization of new algorithms. Such a development will facilitate also the manipulation of quantum registers with an increasing number and complexity of the gates as required, for instance, by various (active or passive) error correction schemes. Other possible extensions refer to the implementation of different *measures* into the FEYNMAN program in order to control the dynamics, entanglement, or the decoherence of quantum algorithms if ‘realized’ by different physical (qubit) systems. Several

of such measures have been implemented already and are currently investigated in dependence of the number of qubits. With these and a few further extensions, we therefore hope that the FEYNMAN program will be helpful not only for teaching the basic elements of quantum computation and quantum information theory but might find its way also into the daily research work in the future.

Appendix A. Auxiliary commands

As explained in Section 3, there are a number of general terms and associated data structures which occur rather frequently in the input and output of the main procedures of the FEYNMAN program. These structures refer, for instance, to the notion of a quantum register or quantum operator and are utilized as the central building blocks of the program. To facilitate the handling of these data structure (i.e. the communication with and within the various procedures), here we first describe all auxiliary procedures as discussed in Section 3.2. Although several tests are made on the input of these commands, they usually return their parameters *unevaluated* and, hence, mainly serve as a *storage* in order to keep the relevant information together in the simulation of the N-qubit systems.

- **cbs([n], k)**
  Auxiliary procedure to represent the *computational basis state* $|k\rangle_{\rm dec}$ within an n-qubit basis in decimal notation.
  Output: An unevaluated call to cbs() is returned.
  Argument options: ([$n_1, n_2, \ldots$], $k_1, k_2, \ldots$) to represent a *composite* computational basis state $|k_1, k_2, \ldots\rangle_{\rm dec}$, where n_i refers to the number of qubits in the ith subspace.
  *([$n_1, n_2, \ldots$], $k_1, k_2, \ldots$, "simplify") to evaluate the composite computational basis state $|k\rangle = |k_1, k_2, \ldots\rangle$ and to return the corresponding (tensor) product state cbs([n],k) with $n = \sum_i n_i$ and where the n_i again refer to the number of qubits in the ith subspace.
  *("00..01") to represent a basis state in the convenient *binary* notation; a computational basis state cbs([n],k) in the *decimal* notation is returned.
  Additional information: A basis of n qubits contains the computational basis states $\{|0\rangle, |1\rangle, \ldots, |k\rangle, \ldots, |2^n - 1\rangle\}$. Owing to the definition of a *computational basis*, therefore, the restriction $k_i < 2^{n_i} - 1$ must apply for all i.
  See also: qbit(), qregister().
- **qbit(id, a, b)**
  Auxiliary procedure to represent a qubit with identifier id in terms of its complex amplitudes a and b with respect to the two (computational) basis states $|0\rangle \equiv \binom{1}{0}$ and $|1\rangle \equiv \binom{0}{1}$; i.e. $|\psi\rangle = a|0\rangle + b|1\rangle$.
  Output: An unevaluated call to qbit() is returned.
  Additional information: The identifier id must be of type `name, string,` or `integer`.
  * If both coefficients, a and b, are given numerically they must fulfill the relation $|a|^2 + |b|^2 = 1$.
  See also: qregister().
- **qoperation([$E_1, \ldots, E_n$])**
  Auxiliary procedure to represent a user-defined n-qubit quantum operation $\mathcal{E}(\rho) = \sum_i E_i \rho E_i^\dagger$ in terms of the operation elements (Kraus operators) E_i.
  Output: An unevaluated call to qoperation() is returned.
  Argument options:
  Additional information: In the present version, the procedure qoperation() is not fully supported by the program. In general, the same parameter lists (argument options) as for the command Feynman_quantum_operation() can be used here. The details of this command, however, will be explained only in a forthcoming contribution.

* Providing a general data structure for different types of quantum operations, no evaluation is associated with a call to this auxiliary procedure. This applies, in particular, for all pre-defined quantum operations; instead, it is mainly utilized in order to simplify the definition of quantum circuits (in a later step of the program development) and to improve the readability of the code.
* The (explicit) representation of the corresponding operation elements is obtained by means of the main procedure Feynman_quantum_operation().
See also: qoperator()

- **qoperator(n, "X", [k])**
Auxiliary procedure to represent (the distributed form of) the pre-defined single-qubit quantum operator X acting on the kth qubit in an n-qubit system.
Output: An unevaluated call to qoperator() is returned.
Argument options: (n, U, $[i_1, \ldots, i_k]$) to represent the distributed form of an user-defined matrix operator U acting on the k qubits $i_1, \ldots, i_k$ of an n-qubit system.
* The same parameter lists (argument options) as for Feynman_quantum_operator() can be used here.
Additional information: Providing a data structure for a large set of quantum operators, no evaluation is associated with a call to this procedure. The explicit matrix representation of the corresponding quantum operator is obtained by means of the main procedure Feynman_quantum_operator().
See also: qoperation(), Feynman_quantum_operator().
- **qregister(id, qbit$_1$, qbit$_2$, ..., qbit$_n$)**
Auxiliary procedure to represent an n-qubit quantum register with identifier id in terms of its individual qubits $\text{qbit}_1, \text{qbit}_2, \ldots, \text{qbit}_n$ (which implies that the quantum register represents a product state initially).
Output: A quantum register qregister(id, n, V) is returned where n is the number of qubits and V a 2^n-dimensional state vector.
Argument options: (id, n, V) to represent an n-qubit quantum register where V is a valid 2^n-dimensional state vector.
*(id, n, M) to represent an n-qubit quantum register where M is a valid $2^n \times 2^n$ density matrix $\rho = \sum_{i,j=0}^{2^n-1} c_{ij}|i\rangle\langle j|$.
Additional information: The identifier id must be of type `name, string,` or `integer`, respectively.
* A quantum register can contain any number of qubits. For the case of n qubits in a product state (i.e. if the qubits are not entangled), a 2^n-dimensional state vector V is generated as the tensor product of the n separate single-qubit input states $|\psi_k\rangle = c_0^k|0\rangle + c_1^k|1\rangle$:

$$|\Psi_{1,2,\ldots,n}\rangle = |\psi_1\rangle \otimes \cdots \otimes |\psi_n\rangle$$
$$= c_0|00\ldots00\rangle + c_1|00\ldots01\rangle + \cdots + c_{2^n-1}|11\ldots11\rangle = \begin{pmatrix} c_0 \\ c_1 \\ \vdots \\ c_{2^n-1} \end{pmatrix},$$

where $\sum_i |c_i|^2 = 1$.
See also: qbit(), qregister().

Appendix B. Selected commands of the FEYNMAN program

To illustrate the use of the FEYNMAN program, this appendix describes in more detail a few (selected) commands from Table 4. Similar as in Appendix A, these *main* procedures are briefly explained together with their optional arguments and some additional information, following the style of the former *The Maple Handbook* [19]. For the list of arguments, the notation from Appendix A is used in the input and output;

for example, a notation like `...qregister`$_a$`,qregister`$_b$`,...` means that the user may type explicitly `...,qregister(id`$_a$`,n`$_a$`,V`$_a$`),qregister(id`$_b$`,n`$_b$`,V`$_b$`),...` in the parameter list, or first assign these (unevaluated) calls to qregister() to some variables, say `wa` and `wb`, and later only use these variables at input time: `...,wa,wb,...`. A complete list of all the exported commands of the FEYNMAN program can be found in the manual file `Feynman-commands.pdf` which is distributed together with the code.

- **Feynman_operator_function(A, f)**
  Evaluates the operator function $f(A) = \sum_i f(\lambda_i)|a_i\rangle\langle a_i|$ of a normal matrix operator A by using its spectral decomposition.
  Output: A matrix is returned.
  Argument options: (qoperator, f) to apply the operator function f to the matrix A as described by qoperator; a qoperator is returned which contains an explicit matrix representation $f(A)$ of the corresponding operator function.
  *(A, f, [taylor, r]) or (qoperator, f, [taylor, r]) to carry out an rth-order Taylor expansion of $f(A)$ instead of the spectral decomposition; a matrix or qoperator is returned in this case owing to the type of the argument.
  Additional information: The function f must be of type `mathfunc`, e.g., `x -> sin(x)`.
  *The operator A must represent a normal operator with a proper spectral representation, i.e. $A = \sum_i \lambda_i|a_i\rangle\langle a_i|$ where λ_i are the eigenvalues of A and $|a_i\rangle$ the eigenvectors.
  See also: qoperator(), Feynman_quantum_operator().
- **Feynman_operator_type(A, *hermitian*)**
  Returns `true` if the matrix operator A is hermitian, or `false` if this is not the case.
  Output: A Boolean value of either `true` or `false` is returned, or `FAIL` otherwise.
  Argument options: (qoperator, *hermitian*) to determine the same for the matrix operator as described by qoperator.
  *(A, *normal*) or (qoperator, *normal*) to return `true` if the matrix operator A (or qoperator) is a normal operator, or `false` if this is not the case.
  *(A, *positive*) or (qoperator, *positive*) to return `true` if the matrix operator A (or qoperator) is a positive operator, or `false` if this is not the case.
  *(A, *positive_definite*) or (qoperator, *positive_definite*) to return `true` if the matrix operator A (or qoperator) is a positive definite operator, or `false` if this is not the case.
  *(A, *unitary*) or (qoperator, *unitary*) to return `true` if the matrix operator A (or qoperator) is a unitary operator, or `false` if this is not the case.
  Additional information: The procedure returns `FAIL` if, for the operator A, the given property cannot be determined uniquely, e.g., if two symbolic expressions cannot be recognized to be equivalent due to an unsuccessful internal simplification.
  *For matrix operators with 'numerical' elements, it may happen that a Boolean value *true* is returned for some particular property even if MAPLE's internal procedure returns `false`. In this procedure, small deviations are accepted for a given property by rounding the numerical values to an internal accuracy of `Digits - 2` digits. Note that this threshold (on the accuracy of the numerical results) depends on the current setting of MAPLE's `Digits` variable.
  *A positive operator A is defined to be an operator whose expectation value $\langle v|A|v\rangle$ is a real and non-negative number for all $|v\rangle$.
  *It can be shown that any positive operator is also hermitian and has a *diagonal representation* $\sum_i \lambda_i|i\rangle\langle i|$, with non-negative eigenvalues λ_i.
  * An operator A is called *positive definite* if $\langle v|A|v\rangle > 0$ for all $|v\rangle \neq 0$.
  *An operator A is said to be *normal* if $A^\dagger A = AA^\dagger$. An operator is normal if and only if it can be diagonalized with respect to some orthonormal basis (spectral decomposition theorem).

*An operator A is said to be *hermitian* (or self-adjoint) if $A^\dagger = A$; obviously, any hermitian operator is also normal.

*A operator A is said to be *unitary* if $A^\dagger A = AA^\dagger = I$; hence, any unitary operator is also normal.

See also: Feynman_quantum_operator().

- **Feynman_product("inner", qregister$_l$, qregister$_r$)**
  Calculates the inner product $\langle \text{qregister}_l | \text{qregister}_r \rangle$, known also as the scalar product of the two quantum states.
  Output: An expression or complex number is returned.
  Argument options:
  (i) **Inner products (or scalar products)**
  ("scalar", qregister$_l$, qregister$_r$) to calculate the same product; the two strings `"inner"` and `"scalar"` always refer to the same definition of the product.
  *("scalar", v, w) to calculate the scalar product for the two (column) vectors v and w.
  (ii) **Kronecker products (or tensor products)**
  ("Kronecker", v_1, v_2) to calculate the Kronecker product $v_1 \otimes v_2$ of the vectors v_1 and v_2 with dimensions m and n, respectively; an $(m \times n)$-dimensional vector is returned.
  *("Kronecker", A, B) to calculate the Kronecker product $A \otimes B$ of the operators A and B if given in terms of two $(m \times n)$ and $(p \times q)$ matrices; an $(mp \times nr)$ matrix is returned.
  *("Kronecker", qoperator$_A$, qoperator$_B$) to calculate the Kronecker product $A \otimes B$ of the two matrix operators A and B as described by qoperator$_a$ and qoperator$_b$, respectively; a qoperator is returned in this case which contains the explicit matrix representation of the operator product.
  *("Kronecker", qregister$_l$, qregister$_r$) to calculate the Kronecker product of the two states as described by the (m-qubit) qregister$_l$ and the (n-qubit) qregister$_r$, respectively; an $(m+n)$-qubit qregister is returned. At output, the qregister contains a state vector representation if both quantum registers were given by state vectors at input time, and a density matrix representation otherwise, cf. qregister().
  *("Kronecker", op-state$_1$, op-state$_2$, ...) to calculate the Kronecker product op-state$_1 \otimes$ op-state$_2 \otimes \cdots$ of two or more matrix operators, density matrices or qregisters; all op-state's must be of the same type (i.e. `Matrix`, `qoperator` or `qregister`). A `Matrix`, `qoperator` or `qregister` is returned depending on the type of the arguments; in the case of qregisters, the state vector representation is used at output time only if all the qregisters at input contained state vectors (and no density matrix representations).
  *("Kronecker_power", A, r) to calculate the rth power $A^{\otimes r}$ of the $((p \times q)$ matrix operator A; a $p^r \times q^r$-dimensional matrix is returned.
  *("Kronecker_power", qoperator, r) to calculate the rth power of the matrix operator as described by qoperator; a qoperator is returned which contains the explicit matrix representation of the product.
  *("Kronecker_power", qregister, r) to calculate the rth power of the state as described by the p-qubit qregister. Depending on the representation of the quantum state in qregister, a qregister in state vector or density matrix representation is returned.
  *("Kronecker_power", "Hadamard", n) to calculate the Hadamard transform of n qubits, $H^{\otimes n}$, using the algebraic formula

$$H^{\otimes n} = \frac{1}{\sqrt{2^n}} \sum_{x,y} (-1)^{x \cdot y} |x\rangle\langle y|.$$

  (iii) **Outer products**
  ("outer", qregister$_v$, qregister$_w$) to calculate the outer product $|v\rangle\langle w|$ of the two state vectors as described by the (n-qubit) quantum registers qregister$_v$ and qregister$_w$, respectively; a qregister containing a $(2^n \times 2^n)$-dimensional matrix is returned (not necessarily a valid state!).
  *("outer", v, w) to calculate the outer product of two n-dimensional (column) vectors. An $n \times n$ matrix is returned.

(iv) **Hadamard product**
("Hadamard", A, B) to calculate the Hadamard product $A \circ B$ of the two matrices with the (same) dimension $(m \times n)$. The Hadamard product is defined as the entrywise product: $(A \circ B)_{i,j} = A_{i,j} B_{i,j}$ and, hence, is a submatrix of the Kronecker product $A \otimes B$. An $m \times n$ matrix is returned.

(v) **Trace products or Hilbert–Schmidt products**
("trace", A, B) or ("Liouville", A, B) to calculate the trace (or Hilbert–Schmidt or Liouville) inner product $(A, B) = \mathrm{Tr}(A^\dagger B)$. An expression or a complex number is returned.

(vi) **Commutators and anticommutators**
("commutator", A, B) to calculate the commutator $[A, B] = AB - BA$ for two quadratic $(n \times n)$ matrix operators A and B; an $n \times n$ matrix is returned.
*("anticommutator", A, B) to calculate the anticommutator $\{A, B\} = AB + BA$ for two quadratic $(n \times n)$ matrix operators A and B; an $n \times n$ matrix is returned.

Additional information: In finite-dimensional (complex) vector spaces, the *inner product space* and the *Hilbert space* are the same; these notations are therefore often used as synonyms.

See also: Feynman_quantum_operator(), qoperator, qregister.

References

[1] P.W. Shor, SIAM J. Sci. Statist. Comput. 26 (1997) 1484.
[2] L.K. Grover, Phys. Rev. Lett. 79 (1997) 325.
[3] J. Wallace, CASYS, Internat. J. Comput. Anticipatory Syst. 10 (2000) 230–245.
[4] H. De Raedt, K. Michielsen, quant-ph/0406210, 2004.
[5] *QuCalc* by P. Dumais, http://library.wolfram.com/infocenter/MathSource/657/;
qmatrix by T. Felbinger, http://library.wolfram.com/infocenter/MathSource/1893/.
[6] J. Preskill, Lecture Notes on Quantum Information and Quantum Computation, Physics 219, California Institute of Technology, 1998.
[7] M.A. Nielsen, I.L. Chuang, Quantum Computation and Quantum Information, Cambridge University Press, Cambridge, 2000.
[8] G. Weihs, T. Jennewein, C. Simon, H. Weinfurter, A. Zeilinger, Phys. Rev. Lett. 81 (1998) 5039.
[9] R. Jozsa, N. Linden, Proc. Roy. Soc. London Ser. A 459 (2003) 2011.
[10] V. Vedral, M.B. Plenio, Phys. Rev. A 57 (1998) 1619.
[11] D. Bruß, J. Math. Phys. 43 (2002) 4237.
[12] K. Blum, Density Matrix Theory and Applications, second ed., Plenum Press, New York, 1996.
[13] C.H. Bennett, IBM J. Res. Develop. 17 (1973) 525.
[14] A. Barenco, C. Bennett, R. Cleve, D. DiVincenzo, N. Margolus, P. Shor, T. Sleater, J. Smolin, H. Weinfurter, Phys. Rev. A 52 (1995) 3457.
[15] M.J. Bremner, C.M. Dawson, J.L. Dodd, A. Gilchrist, A.W. Harrow, D. Mortimer, M.A. Nielsen, T.J. Osborne, Phys. Rev. Lett. 89 (2002) 247902.
[16] H.-P. Breuer, F. Petruccione, The Theory of Open Quantum Systems, Oxford University Press, 2002.
[17] K. Kraus, States, Effects and Operations: Fundamental Notions of Quantum Theory, Springer, 1983.
[18] J. Kempe, D. Bacon, D.A. Lidar, K.B. Whaley, Phys. Rev. A 63 (2001) 042307.
[19] D. Redfern, The Maple Handbook, Springer, New York, 1996.
[20] G.F. Viamontes, I.L. Markov, J.P. Hayes, in: Proc. of Quantum Computation and Information, SPIE Defense and Security Symposium, 2004.