Computer Physics Communications 185 (2014) 1697–1718

Contents lists available at ScienceDirect

Computer Physics Communications

journal homepage: www.elsevier.com/locate/cpc

The FEYNMAN tools for quantum information processing: Design and implementation☆

S. Fritzsche*

Helmholtz-Institut Jena, Fröbelstieg 3, Germany
Theoretisch-Physikalisches Institut, Friedrich-Schiller-Universität Jena, Max-Wien-Platz 1, D-07743 Jena, Germany

ARTICLE INFO

Article history:
Received 6 August 2013
Received in revised form
10 November 2013
Accepted 2 February 2014
Available online 11 February 2014

Keywords:
Entanglement and entropy measures
Quantum information processing & theory
Quantum measurements
Quantum operations and operators
Parametrization of quantum states
Quantum tomography
Teaching quantum information science
Tools for quantum simulations

ABSTRACT

The FEYNMAN tools have been re-designed with the goal to establish and implement a high-level (computer) language that is capable to deal with the physics of finite, n-qubit systems, from frequently required computations to mathematically advanced tasks in quantum information processing. In particular, emphasis has been placed to introduce a small but powerful set of *keystring*-driven commands in order to support both, symbolic and numerical computations. Though the current design is implemented again within the framework of MAPLE, it is general and flexible enough to be utilized and combined with other languages and computational environments. The present implementation facilitates a large number of computational tasks, including the definition, manipulation and parametrization of quantum states, the evaluation of quantum measures and quantum operations, the evolution of quantum noise in discrete models, quantum measurements and state estimation, and several others. The design is based on a few high-level commands, with a syntax close to the mathematical notation and its use in the literature, and which can be generalized quite readily in order to solve computational tasks at even higher degree of complexity.

In this work, I present and discuss the (re-design of the) FEYNMAN tools and make major parts of the code available for public use. Moreover, a few selected examples are shown and demonstrate possible application of this toolbox. The FEYNMAN tools are provided as MAPLE library and can hence be used on all platforms on which this computer-algebra system is accessible.

Program summary

Program title: FEYNMAN

Catalogue identifier: AESG_v1_0

Program summary URL: http://cpc.cs.qub.ac.uk/summaries/AESG_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.: 22626

No. of bytes in distributed program, including test data, etc.: 859217

Distribution format: tar.gz

Programming language: Maple 15 and 16.

Computer: Any computer capable of running the Maple package.

Operating system: Suse and Ubuntu Linux.

Typical time and memory requirements: Most commands respond promptly or require less than a few seconds on a standard processor if invoked with quantum states and registers of four or less qubits. In addition to the memory for using the Maple machinery, about 5–20 MB or working memory is typically sufficient to deal with standard (floating-point) computations. When working with symbolic expressions or some large number of qubits ($n \gtrapprox 5$), however, the requirements of cpu time and memory depend

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

* Correspondence to: Helmholtz-Institut Jena, Fröbelstieg 3, D-07743 Jena, Germany. Tel.: +49 3641 947606; fax: +49 3641 947602.
E-mail address: s.fritzsche@gsi.de.

http://dx.doi.org/10.1016/j.cpc.2014.02.003

critically on the size and complexity of the expressions involved. Although Maple's assume feature may sometimes help reduce the complexity of symbolic expressions, more often than not, only numerical computations are feasible for sizeable expressions or a larger number of qubits. Since most commands can be readily applied to quite different computational scenarios, no general *scaling* rule can be given for the requested cpu time or memory.

Classification: 4.15.

Nature of problem:
Quantum information processing and quantum control require a careful preparation and analysis of quantum states. For such an analysis and for implementing new protocols, various computational tasks need to be performed frequently, such as the construction, manipulation and estimation of quantum states, the display of quantum amplitudes and probability distributions, the evaluation of quantum measures and quantum operations, the decomposition of operators into different representations, or the simulation of quantum measurements, to name just a few.

Solution method:
To support a large number of computational tasks in dealing with n-qubit quantum states and quantum registers, a high-level (computer-) language is designed and implemented within the framework of the computer-algebra system Maple. This design is based on a set of *keystring*-driven commands in order to provide a flexible, interactive and user-friendly environment for quantum computations on finite n-qubit systems. The present design provides high-level commands (procedures) that can be utilized in order to simulate reversible and irreversible quantum computations. Though the use of Maple has been found versatile to access a wealth of (additional) in-*built* mathematical functions, the re-design of the FEYNMAN tools is general and flexible enough to be implemented (or combined) with other languages and computational environments.

Restrictions:
Restrictions in the manipulation of n-qubit quantum states mainly arise from the complexity and stability of certain algebraic and numerical computations. For example, the symbolic analysis of noise models (as defined by means of the known Kraus operators) often leads to large expressions that slow down all subsequent computations. Owing to the exponential increase of complexity with the number of qubits involved, many computational tasks are limited to about 4–5 qubits on a standard processor, although some larger number of qubits might become feasible by using Maple's parallel features or some link to specially designed code in Matlab, C or Fortran.

Unusual features:
The FEYNMAN tools [1] have been re-designed within the framework of Maple to provide a small but very powerful set of *keystring*-driven commands for symbolic and numerical simulations of (finite) n-qubit systems. Emphasis was placed especially on establishing a set of (about 15–20) high-level commands that are suitable to deal with most common tasks from quantum information theory, and which can be applied also interactively. Whenever possible, both representations of quantum registers in terms of their state vectors and/or density matrices are equally supported by the program, and with no other restrictions than those given by the memory and processor resources of the computer.

All commands of the FEYNMAN tools are organized in an hierarchical order and can be utilized as language elements in order to perform tasks at even higher level of complexity. Moreover, the design and syntax of this toolbox may serve as a model to apply computer-algebraic techniques to other fields in science and technology.

Additional comments:
For the original FEYNMAN programs see http://cpc.cs.qub.ac.uk/summaries/ADWE_v1_0.html [1] to http://cpc.cs.qub.ac.uk/summaries/ADWE_v5_0.html

Running time:
About 2 min on a standard laptop for all test cases below.

References:
[1] T. Radtke and S. Fritzsche, Comput. Phys. Commun. 173 (2005) 91; ibid. 181 (2010) 440.

1. Introduction

During the last two decades, quantum information processing (QIP) has developed rapidly into a new and well-established field of research. This field comprises many different subjects from physics, mathematics and computer science that were worked out independently and are nowadays combined in order to harness the fascinating properties of the quantum world. Its goal is to achieve more efficient (quantum) computations and communications by using non-classical states of matter. In particular, a number of new protocols and architectures have been realized during recent years, such as teleportation [1,2], secure communication [3,4], distributed quantum processing [5], and many more, and this may pave the way for more efficient QIP with no classical counterpart in the future.

Despite the large number of successful experiments, however, many open questions and (experimental) challenges remain with regard to decoherence and our (present) understanding of how quantum correlations, entanglement or the non-locality of quantum states can be utilized in order to make most out of the quantum nature of matter; cf. Ref. [6] for further details. To advance our knowledge about this topic, much algebraic and numerical work has been performed and has led to a number of sub-communities in QIP with quite

Table 1
Selected key topics from QIP in studying n-qubit quantum systems and states. For each of these topics, a few computational requirements and tasks are listed here.

Topic	Computational requirements and tasks
Decoherence & dissipation	Evaluation of quantum operations for given quantum states; maps between quantum operations and quantum states; computation of super operators, Choi matrices and Kraus operators.
Non-locality	Evaluation of probabilities and inequalities, such as the CHSH inequality by Clauser, Horne, Shimony and Holt [12]; computation of (mutual) information measures between pairs of quantum states.
Quantum algorithms	Quantum phase estimation; evaluation of the quantum Fourier and Hadamard transform; simulation of quantum walks, etc.
Quantum correlations	Computation of quantum measures of various kinds.
Quantum cryptography & key distribution	Noise evolution in quantum key distribution; information flow in quantum-storage models; time-evolution of quantum entanglement.
Quantum entanglement	Optimization of quantum measures over some complete set of pure states or density matrices; parametrization of matrices of various kind.
Quantum error correction	Construction of stabilizers and group generators; projective measurements in complex bases; frequent use of quantum transformations.
Quantum measurements	Random state generation; measurement-based quantum computations; simulation of Bell-state measurements and measurement outcomes; generalized POVM and weak measurements.
Quantum networks & teleportation	Implementation of teleportation protocols in different basis sets; analysis of multipartite entangled state; computation of distance measures.
Quantum simulations	Decomposition of gate operations; Hamiltonian dynamics in external fields; behavior of trapped ions and spin-chains.
Quantum state distillation	Conversion of imperfectly entangled states into (a smaller number of) maximally entangled states; entanglement purification.
Quantum state & process tomography	Determination of mutually unbiased basis (MUB); computation of probability distributions; protocols for quantum state estimation.

different techniques and interests. The common (mathematical) background of these communities makes it however desirable to have computational tools available which facilitate the handling of such computational techniques but stays otherwise as close as possible to the notations as applied within the literature.

While several such tools and quantum simulators *do exist* already, either as described in the literature [7,8] or accessible from the web, many of them are devoted to just a (very) few tasks and with a design that is incompatible with applications of some larger size. A good number of such quantum simulation tools are listed in Ref. [9]. Very similar limitations also apply to our previous implementation of the Feynman program [10] which was developed in a number of steps but with an (earlier) design that could not be extended so easily [11]. Therefore, a re-design became desirable that enables one to cover quite different topics but with little increase in the complexity of the program. In particular, a small but powerful set of commands (instructions) became desirable that facilitate typical computations as they appear frequently in the simulation of quantum registers.

To pursue this objective, I here present and discuss the re-design of the Feynman tools. The major goal was to work out a high-level language that is capable to support most computational tasks in dealing with finite (n-qubit) systems. Emphasize has been placed especially to define a small but powerful set of *keystring*-driven commands for both, symbolic and numerical computations. While we use Maple again to implement the Feynman tools, the design is general and flexible enough for being implemented also in other languages and environments. In contrast, less attention has been paid at present to extending much the size of these tools, even though a number of new and useful features have been added to the Feynman tools.

In the next section, I first discuss the requirements in simulating finite n-qubit systems (on classical computers), including typical computations in this field as well as some of the successes and limitations of previous implementations. A short account is given especially to the difficulties which one needs to overcome in practice as well as to some *pros* and *cons* in using computer-algebra systems (CAS). Section 3 then describes the re-design and implementation of the Feynman tools. Apart from a brief overview about these tools, I shall outline here the basic data structures and the (generic) design of the main procedures. While all main procedures are at least summarized and tabulated in this long write-up below, all further information about these user-accessible commands is given in a manual that is provided together with the Feynman tools. The use of basic data structures hereby help explain the syntax and design of these tools without that the description becomes overloaded. Section 3 also contains a few technical remarks as well as comments on the installation and distribution of the code. To demonstrate the power and flexibility of the present re-design, Section 4 later displays a few advanced examples, including the characterization of single-qubit channels or the swapping of entanglement via some *prior* or *delayed* Bell-state measurement. Finally, a summary and brief outlook is given in Section 5.

2. Simulation of n-qubit systems

2.1. Typical and frequent requirements

While QIP has already enhanced our understanding of the quantum world, many further questions need to be explored *before* quantum systems can be utilized eventually for non-trivial computations. These questions concern, for example, the role of entanglement as a crucial but very fragile resource in QIP and its effect upon the efficiency of quantum algorithms, or how the entanglement of some state can be protected against decoherence in some particular quantum protocols and its practical realization. Further interest in working with n-qubit systems arises from the evolution of quantum states and registers, the characterization of quantum correlations or the search for efficient quantum state and process tomography, to mention just a few. Table 1 lists several key topics from quantum information science together with some of their computational requirements that frequently occur in dealing with the classification and manipulation of n-qubit systems.

Other applications in this research field refer to the definition and manipulation of quantum states, the study of (quantum) noise, or the evaluation and parametrization of quantum operations. Analogue to classical computers, n-qubit systems are typically handled by

means of quantum registers which are either in a *pure* or *mixed* state and, hence, may be represented by state vectors or density matrices, respectively. While the density matrix formalism is known to be very powerful, it is often desirable to just work with state vectors as long as no mixtures occur or if these mixtures can be approximated by such vectors reasonably well. Moreover, quantum operations, i.e. completely positive and trace-preserving maps (CPT maps), need to be treated efficiently in order to describe general (unitary or nonunitary) transformation of quantum states and, especially, to simulate decoherence effects in quantum registers. For these reasons, quantum registers, quantum operators and quantum operations are three important (building) blocks that need to be handled very easily in almost all fields of QIP.

Other frequent tasks refer to the evolution of quantum measures, the composition and decomposition of quantum operators or the optimization of quantum measures with regard to a complete set of states. For the latter, an efficient parametrization of quantum registers and other objects, such as hermitian and unitary matrices, is required and often provides the only basis for heuristic studies on the evolution of entanglement. Below, we shall describe how these and many other tasks can be accessed quite easily by means of the FEYNMAN tools.

2.2. Previous implementations. Difficulties to overcome

Over the years, the great promise and advancement of QIP has lead to a large number of implementations which (often) deal with just one particular or a very few computational requests mentioned above. For example, various worksheets have been worked out by different people in order to implement and demonstrate the performance of various quantum algorithms, including those by Deutsch–Josza [13], Shor [14] or Grover [15]. Indeed, it appears rather difficult today to *oversee* all research activities that were undertaken previously or are still under active work at present. A rather comprehensive list of quantum simulators and software tools for QIP is listed on the web-page [9] but with many having the status *finished* or *unknown*.

Despite of all the effort at various places worldwide, however, no general quantum simulation software is still available or has been user-friendly enough in order to attract a large number of users. Especially many more advanced topics of QIP have hardly been addressed at all by proper software tools including, for example, the evaluation of quantum measures, the time and entanglement evolution of quantum states under decoherence (noise), or quantum state and process tomography. Most software tools that are accessible from the web today deal instead with *ideal* quantum operations and their subsequent application in quantum algorithms or protocols. In addition, many of these tools only allow numerical simulations, such as QUBIT4MATLAB [7], LIBQUANTUM [16] or QWALK [17], and with the latter two being based on some C/C++ programs and libraries.

Similar to many of these software projects, we started developing the FEYNMAN tools almost ten years ago [10]. Our goal at that time was to support different computational techniques from QIP by means of computer algebra and to help the user with both, symbolic as well as numerical computations. Several steps were undertaken over the years in order to incorporate, beside of some basic data structures, also quantum measures [18], the parametrization of some frequently occurring objects [11], or even quantum measurements [19]. Although emphasis was placed on providing a flexible and easily extendible framework, the initial design of the FEYNMAN tools, i.e. the organization and syntax of the various procedures, was found less suitable for maintaining and enlarging the project. Especially the steadily growing number of procedures, their improper naming and the different combination of keywords and keystrings for specifying some particular action or the (often varying) list of parameters made further improvements on the code very cumbersome and a re-design advisable.

Indeed, many implementations which are presently available from the web focus upon a few selected tasks and are overloaded with a large number of keywords or flags. Such an 'piece-by-piece' implementation of additional functions makes the syntax of a software project often cryptic and prone to errors, especially if it is employed only occasionally by the user. In order to overcome this *weakness* in having a large but rather inconsistent number of software pieces, a Quantum Computation Language (QCL) was developed as a high-level and architecture independent programming language for quantum computers [20], and with a syntax derived from the classical procedural languages, such as C or Pascal. Various general goals were defined for this QCL project in order to support, for example, the implementation and simulations of quantum algorithms (including the classical components) into just *one* consistent formalism. Until the present, however, no software tools are yet available that realize this formal approach.

Today, instead, general CAS systems have a great potential to deal with (mathematical) formalisms that are well established theoretically but which may become cumbersome, if applied to complex systems. Apart from their high reliability, most of these systems enable one to specify the precision, to parallelize the code (rather automatically) or to use features for grid computing that are built into the standard. These features make the general CAS, such as MAPLE and MATHEMATICA, very suitable to work out new ideas and concepts. Although some of these features must be bought with a rather large overhead and slow-down of the computations (when compared to other compiled languages), the advantages of general CAS dominate and have given the impact to develop the FEYNMAN tools within MAPLE. Below, we explain how our experience with the previous implementation helped to overcome most of the difficulties mentioned here and to provide a toolbox that facilitates a large number of tasks, including the definition, manipulation and parametrization of quantum states, the evaluation of quantum measures and quantum operations, the evolution of quantum noise in discrete models, quantum measurements and state estimation, and many others.

3. Redesign and implementation of the FEYNMAN tools

3.1. Brief overview. General design principles

As outlined before, the FEYNMAN tools have been developed for simulating the physics and behavior of n-qubit systems. Hereby, our main intention is to support general concepts and manipulations in quantum information science, while (considerably) less attention was paid to the physical realization, such as ion traps, nuclear magnetic resonances, or many other experimental techniques that are nowadays available. Beside of the definition and manipulation of quantum states, the FEYNMAN tools support a large number of computational tasks, such as the evaluation of quantum measures and quantum operations, the display of quantum amplitudes and probability distributions, the decomposition of quantum operators into different representations, or the simulation of quantum measurements, to tell just a few.

At present, the FEYNMAN tools are implemented within the framework of MAPLE. Within this CAS environment, a (hierarchical) set of procedures has been defined for providing data structures as well as commands that operate upon these structures. In particular,

Table 2
Data structures of the FEYNMAN tools in alphabetic order. These structures help keep relevant information together about several frequently occurring entities. They are often used for the input and output of the main commands.

Procedure	Represents an instance of
cbs()	A *computational basis state* of either a single- or composite multi-qubit system.
qbasis()	An (orthogonal) n-qubit basis, either in string notation or in terms of its up to 2^n basis states.
qbit()	A qubit, $\|\psi\rangle = a\,\|0\rangle + b\,\|1\rangle$, in terms of its complex amplitudes a and b.
qoperation()	A pre- or user-defined (multi-qubit) quantum operation in the operator-sum formalism, either in string notation or in terms of its operation elements.
qoperator()	A (single- or multi-qubit) quantum operator, either in string notation or in terms of its explicit matrix representation.
qregister()	The (pure) state or density operator of a n-qubit quantum system.

emphasis was placed to define (i) a small but powerful set of commands and to use (ii) basic data structure (sometimes referred to as auxiliary procedures below) in order to keep relevant information together. Within MAPLE, these auxiliary procedures are just used as data 'containers' as they are otherwise provided by records or derived data in most other languages.

When compared with its previous implementation [11,19], much attention in the re-design of the FEYNMAN tools was given to define a set of (about 15–20) *keystring*-driven commands in order to support a high-level description of many computational tasks from QIP. All these commands can be invoked interactively as well as language elements for building up some higher level of complexity. In practice, only a rather small effort will often be required in order to extend the program and adapt it to some user-specific needs. However, while just 15–20 commands are *seen* by (and exported to) the users, the whole program currently comprises more than 150 procedures from which most remain hidden.

Indeed, a primary goal in re-organizing the code was to reduce the total number of commands as they are accessible by the user. This was achieved by means of *generic* procedures and by collecting similar functions and tasks into the same command. All of these high-level commands usually refer to some (ambiguous) verb, such as *compute, convert, define* or *evaluate*, and which expect a `"keystring"` as their first argument in order to specify the particular action in further detail. This simple and consistent design makes the performance of even complex tasks quite *descriptive* and also facilitates the implementation of the code as the generic (main) commands only deal with the handling and distribution of the input data.

For example, the (generic) command Feynman_compute() enables one, for a given matrix or quantum register, to compute the adjoint matrix, eigenvalues and eigenvectors, the rank, partial traces and transpositions and many other entities, and these actions are readily determined by the corresponding keystrings `"adjoint"`, `"eigenvalues"`, `"eigenvectors"`, `"rank"`, `"partial trace"` or `"transpose: partial"`, respectively. The present re-design of the FEYNMAN tools therefore comprises many features into some single command that were previously scattered over several procedures, and often with names that were not easy to remember. A short list of all supported tasks (keystrings) is obtained from each command if invoked *without* arguments, e.g. Feynman_compute(). For many keystrings, moreover, different types of input parameters are supported and then result in output of some appropriate type. For all further details and functions of the FEYNMAN tools, we refer the reader to a manual, `Feynman-manual.pdf`, that is provided with the code.

In practice, the re-design of the FEYNMAN tools has been built upon a few general principles that may help simplify the use and maintenance of the code and will hopefully ensure a long life cycle. These principles are based on our previous experience with the theory of angular momentum [21,22] and refer to the use of:

Keystrings: are utilized to concisely describe the action of each command. With each keystring, one or several lists of parameters are associated and facilitate the use of the code. As mentioned before, all presently supported keystrings are listed on screen if the command is called without arguments.

Few but powerful commands: with a clear description and consistent use of parameters. Despite some computational overhead in using such generic names, this principle improves the readability of the code and its interactive use; it also facilitates the handling and maintenance of the program.

Use of basic data structures: Quantum operations and quantum registers are just two important terms in QIP that are used very frequently to implement quantum algorithms and protocols. In the FEYNMAN tools, the procedures qoperator() and qregister() are used – along with a few others – to keep the relevant information together and to facilitate the input and output of many commands.

Improved readability of the code: For large software projects, a good readability often appears more important for a long life cycle and success of some code than efficiency alone. In the present design of the FEYNMAN tools, therefore, attention was paid to a simple structure and readability of the code, while the efficiency was sometimes treated secondary.

From these general rules, it is seen quite easily how further tasks and functions will be added to the code without that the current implementation is affected so much. Therefore, further functions might be incorporated also by the user or due to the request from outside.

3.2. Data structures: keeping relevant information together

Properly *derived* data types are essential for every software applications. For the simulation of n-qubit quantum systems, especially quantum registers, quantum operators and quantum operations are three important building blocks on which manipulations are done in order to handle different tasks in quantum information theory. In the FEYNMAN tools, these building blocks are realized by the procedures qregister(), qoperator() and qoperation(). They help to keep relevant information together and facilitate the data handling and communication within the program. These auxiliary procedures always return *unevaluated*.

Table 2 displays all data structures of the FEYNMAN tools in alphabetic order. For example, a call of qregister(n,state) defines the state of an n-qubit quantum register, either in terms of a (normalized) 2^n state vector or a $2^n \times 2^n$ density matrix. If such a qregister() occurs in the program, a test is made only for n, the number of qubits, to be consistent with the dimension of the associated vector or matrix but, otherwise, this procedure returns *unevaluated*. Similarly, a qoperator() defines a quantum operator either in keystring notation, and

Table 3
Main commands of the FEYNMAN tools in alphabetic order. Apart from a brief description of each command, we here list the central tasks and subtopics that can be handled by these procedures by using proper keystrings.

Command	Short description & classification into subsections
Feynman_apply()	Applies a quantum operator or quantum operation to the state of a given qregister().
Feynman_compute()	Computes various quantities for given quantum states, operations or transformations; this includes the computation of: (a) Adjoint quantum states, operators and operations; (b) Eigenvectors and eigenvalues; (c) Norm and normalized states; (d) Flatten/unflatten of matrices; (e) Ranks; (f) Traces and partial traces; (g) Expectation values and variances; (h) Matrix transpose and partial transpositions of a matrix; or (i) Miscellaneous.
Feynman_convert()	Converts different representations of numbers, quantum states or quantum operations into each other; this includes the conversion: (a) between integers and binaries; (b) cbs() into qregister(); and (c) between state vectors and density matrices.
Feynman_decompose()	Carries out various decompositions of a given quantum operator including: (a) Euler decompositions; (b) Kronecker decompositions; (c) Pauli or gate decompositions; (d) Polar and singular-value decompositions; (e) Schmidt decompositions; and (f) Spectral decompositions.
Feynman_define()	Defines various states that occur frequently in the literature or textbook examples including: (a) Linear combinations and mixtures; (b) Diagonal states $\|+\rangle$, $\|-\rangle$, $\|0\rangle$, $\|1\rangle$, $\|L\rangle$, $\|R\rangle$ and products thereof; (c) Initialization and random states; (d) Predefined states from the literature; (e) Maximally mixed and entangled states; and (f) Graph states.
Feynman_display()	Displays different entities and results in a neat and convenient format including (at present only): (a) Measurement operators and projectors.
Feynman_evaluate()	Evaluates various products and quantities for quantum states, quantum operators or quantum operations, including the evaluation of: (a) Inner products; (b) Kronecker and tensor products; (c) Outer products; (d) Hadamard products; (e) Direct sum; (f) Commutators and anticommutators; (g) Matrix and operator functions; or the (h) Evaluations on quantum operations.
Feynman_generate()	Generates different (lists of) matrices or quantum states including: (a) Permutation matrices; (b) Group generators; (c) Random states and matrices; or (d) Mutually unbiased basis.
Feynman_is()	Tests some given data (structures) and objects for certain properties including tests on: (a) Properties of quantum states and density operators; (b) Properties of quantum operators; (c) Properties of quantum operations; (d) Properties of basis states and bases; (e) Equivalence of given data; or (f) Miscellaneous.
Feynman_measure()	Performs various types of measurements on the state of a given qregister including: (a) Single-qubit projective measurements; (b) Further projective measurements; or (c) Generalized and POVM measurements.
Feynman_measures()	Computes varies measures for one or several quantum states, operators and others; these measures include: (a) Distances and fidelities; (b) Entropy measures; (c) Entanglement and non-locality measures for bipartite systems; (d) Entanglement measures for multipartite systems; or (e) Measures for quantum operations & channels.
Feynman_parametrize()	Generates various quantum states, operators and matrices for a given set of parameters and by applying different parametrization methods. In more detail, this command supports the: (a) Generation of pure states; (b) Generation of mixed states; (c) Generation of unitary matrices; (d) Generation of hermitian matrices; and (e) Miscellaneous.
Feynman_perform()	Performs various (more or less) complex tasks on quantum states. This command is presently still under work.
Feynman_plot()	Plots the state of qubits, quantum states, density matrices, etc. in a nice format; the current features include plots of: (a) State vectors and density matrices.
Feynman_print()	Prints the state vector or density matrix of a quantum register in a neat format.
Feynman_qoperation()	Returns a requested quantum operation in terms of its operation elements; this command distinguishes between: (a) Single-qubit quantum operations; (b) Distributed single-qubit quantum operations; and (c) n-qubit quantum operations.
Feynman_qoperator()	Returns the matrix representation of some specified gate or quantum operator; this command distinguishes between: (a) Single-qubit quantum gates; (b) Two-qubit quantum gates; (c) Three-qubit quantum gates; (d) n-qubit quantum gates; (e) Distributed single-qubit quantum gates; (f) Distributed two-qubit quantum gates; (g) Distributed three-qubit quantum gates; and (h) Distributed n-qubit quantum gates.
Feynman_simulate()	Simulates the outcome of various quantum processing tasks, such as (at present only): (a) Measurement outcomes.
Feynman_transform()	Transforms some given state or matrix into a new (and specified) basis; this command supports the following features: (a) Transform a basis by a given matrix; (b) Transform into the Hadamard basis; (c) Transform or permute the qubit basis; and (d) Transform into a user-given qubit basis.

with a few parameters if appropriate, or explicitly in terms of its matrix representation. The use of keystrings and associated parameters is typically preferred since this form reduces the storage requirements and also improves the readability of the program during interactive work or the debugging of the code. If required, an explicit matrix representation of the (n-qubit) operator can be obtained very easily at any time by calling the command Feynman_qoperator() with the given qoperator() as single argument, cf. Section 3.3. Therefore, the use of such keystrings provides a rather compact notation of quantum operators which can be evaluated later on demand.

An analogue 'couple' of procedures, one providing the data structure and a second for the corresponding main command, also exists for quantum operations and quantum bases. Quantum operations can be utilized to describe general (unitary or nonunitary) transformation of quantum states as they occur for simulating decoherence effects in quantum registers. A quantum operation in the operator-sum representation is handled internally by a qoperation() with some proper keystring notation, or explicitly in terms of its operation elements. Again, an explicit matrix representation of the operation elements can be obtained at any time by invoking the command with the given qoperation() as single argument.

In the present version, moreover, we now added also the data structure qbasis() in order to represent either a pre- or user-defined quantum basis. This structure can comprise, for instance, a mutually unbiased basis as they occur frequently in quantum estimation protocols. In the future, this concept of basic data structures will likely be extended also to define quantum circuits or even whole *protocols* in the FEYNMAN tools in order to make them easier accessible for use and analysis.

3.3. Main commands: manipulation and analysis of quantum states

Table 3 displays all *main* commands of the FEYNMAN tools that are accessible by the user. Whenever appropriate, we here use simple *verbs*, such as apply, compute, convert, decompose, define, …in order to describe in an *active* form of what a command is "going to do" with its given list of parameters. Most commands are divided into a number of subsections in order to embrace different but related computational tasks. The particular purpose of what the main commands will do is typically controlled by a `"keystring"` as the very first argument, and followed by a proper number and type of parameters. As explained before, we make use of such keystrings, in contrast

to the *named* or *unevaluated* variables as usually applied in MAPLE, in order to allow a descriptive use of the main commands that explains itself (more or less). In addition, this design also simplifies the implementation and maintenance of the program.

A "keystring" can contain either a single word, a short phrase, or even further explanations after a (semi-) colon. In the command Feynman_define(), for example, the keystrings "Bell", "Bell: Phi+" or "Bell: diagonal" may be used (beside of many others), in order to 'set-up' the internal representation of the well-known Bell states from the literature and to make them accessible for further manipulations. The distinction of separate subtopics and a list of all keystrings that are currently supported by a given command is printed (to screen), if the procedure is called without parameters. For example:

```
> Feynman_define();

The following 'keystrings' are supported by this command:
-----------------------------------------------------------
  a) Linear combinations & mixtures::       state                    mixture
  b) Diagonal & string states::             string
  c) Initialization & random::              equal                    Walsh-Hadamard      random
  d) Predefined states::                    Bell: Phi+               Bell: Phi-          Bell: Psi+
                                            Bell: Psi-               Bell                Bell diagonal
                                            biseparable              Dicke               GHZ
                                            ground state             Ishizaka            IH
                                            isotropic                Smolin              thermal
                                            W                        Werner
  e) Maximally mixed & entangled states::   maximally entangled      maximally mixed     MEMS
  f) Graph states::                         graph                    graph: chain        graph: ring
```

lists the six subsections (a–f) and the associated keystrings for this command. In practice, the use of keystrings in controlling a command has reduced remarkably the number of optional arguments, and which are now absorbed into the keystring notation.

Here, we need not explain all (main) commands from Table 3 in much detail; all information about the optional argument lists, the corresponding output and further details are described in the manual to the FEYNMAN tools as appended to the code. Instead, I shall outline only the general design and functions of the FEYNMAN tools by giving a short account on just seven commands that help, for instance, compute and inquire about the properties of quantum states or to simulate quantum measurements on them. I also briefly explain here how different quantum measures can be determined for given states and operations.

Feynman_compute(): As seen from Table 3, this command comprises eight (sub-) section to compute various useful quantities for quantum registers, quantum operators or quantum operations. These sections, together with the presently supported keystrings, are listed by:

```
> Feynman_compute();

The following 'keystrings' are supported by this command:
-----------------------------------------------------------
  a) Adjoint states, etc. ::     adjoint
  b) Eigenvectors & values::     eigenvalues          eigenvectors          eigenvectors: normalize
  c) Norm & normalized states::  norm                 trace norm            Hilbert--Schmidt norm
                                 HS norm              normalized state
  d) Flatten & unflatten::       flattened vector     (unflattened) matrix  matrix
  e) Ranks::                     rank
  f) Traces & partial traces::   trace                partial trace         partial trace: subsystems
  g) Expectation values::        expectation value    variance
  h) Matrix transpose, etc.::    transpose            transpose: partial    transpose: column
                                 transpose: raw
  i) Miscellaneous::             simplify             sqrt                  sqrt: Denman
                                 correlation tensor
```

Many features of this command are quite obvious already by reading the keystrings above, and only the possible argument (lists) for the various branches of the code need to be explained in further detail in the manual. Instead of vectors and matrices, also quantum registers and quantum operators may occur as arguments to Feynman_compute(), and they then results in a corresponding output format. Apart from the (straightforward) computation of eigenvalues and eigenvectors, normalization constants, ranks and traces, this command also supports the computation of partial traces and partial transpositions for selected partitionings and dimensions of the subsystems. In addition, Feynman_compute() enables one to calculate the expectation value and variance of an operator with regard to some given quantum state, or to determine the correlation tensor of some (density) matrix as well as various other tasks.

Feynman_is(): This command enables one to 'test' given data for different properties. It extends MAPLE's well-known type facilities towards more complex properties, such as the separability criteria for quantum states or some general properties of quantum operators, operations or basis sets, respectively. This command also help 'test' for the equivalence of data given in different representations. For example, Table 4 lists the separability criteria that are supported by Feynman_is(), and on which the state of some quantum register can be tested for. Again, a brief list of all supported keystrings is given by:

```
> Feynman_is();

The following 'keystrings' are supported by this command:
-----------------------------------------------------------
  a) States & density          rho                      pure                     separable: majorization
              operators::      separable: PPT           separable: reduction     separable: Schmidt
  b) Quantum operators::       anticommuting            commuting                normal
                               positive                 positive definite        unitary
  c) Quantum operations::      collective               completely positive      extremal
                               hermiticity-preserving   permutation invariant    trace preserving
                               unital
  d) Basis states & basis::    MUB                      orthogonal
  e) Equivalence of data::     equal
  f) Miscellaneous::           identity                 ket                      bra
                               linearly independent
```

Table 4
Separability criteria supported by the command Feynman_is(). A Boolean value of `true` is returned if the requested property is fulfilled and `false` otherwise. See the manual for further details on these criteria.

Argument option	To test whether
("separable: Schmidt", …)	the given *pure* state in qregister$_{AB}$ is separable with respect to the specified *bipartite* cut. Although the given qregister() may contain a density matrix, it must describe a pure state.
("separable: majorization", …)	the given n-qubit qregister() is *bipartite* separable into the two parts $[[i_1, \ldots, i_k]$ and $[i_{k+1}, \ldots, i_n]]$ due to the majorization criterion. Here, the qubit indices $1, \ldots, n$ can occur in any order in the two lists.
("separable: PPT", …)	the given (n-qubit) qregister is separable due to the *positive partial transpose* (PPT) criterion, also known as Peres–Horodecki criterion.
("separable: reduction", …)	the given qregister is *bipartite* separable due to the reduction criterion.

In all these cases, a Boolean value of `true` or `false` is returned if the decision about the given property can be made, and `FAIL` otherwise. For testing the equivalence of two vectors, matrices or quantum registers, a user-given *noise threshold* can be given, in addition, in order to account for rounding errors in course of some computation.

Feynman_measure(): This command simulates various types of quantum measurements for the state of a given qregister(). Apart from single-qubit projective measurements along different directions, this command supports projective Bell measurements, the measurement of some observable (hermitian operator) as well as generalized and positive-operator valued measure (POVM) measurements. In the latter two cases, a set of measurement operators or POVM elements must be given explicitly. The list of presently supported keystrings is:

```
> Feynman_measure();

The following 'keystrings' are supported by this command:
-----------------------------------------------------------
  a) Single-qubit, projective::          projective: X      projective: Y           projective: Z
                                         projective: Bx     projective: Bz          projective: Bloch
  b) Further projective measurements::   projective: Bell   projective: observable
  c) Generalized & POVM measurements::   generalized        POVM
```

Typically, a table T is returned for each call to this procedure which has three entries:
`T[outcomes], T[probabilities]` and `T[post_ms]` in order to provide the (possible) outcomes and their associated probabilities and post-measurement states. If no particular outcome was selected during the call to this command, either as *random* choice or by specifying a particular eigenvalue, each of these three entries contain a list of m elements that correspond to the possible measurement outcomes $(1, \ldots, m)$, but in a sequence that depends on the particular installation. If further details about the outcome of the measurement are specified, the entries: `T[outcomes], T[probabilities]` and `T[post_ms]` will refer directly to the corresponding result of the simulated measurement.

Feynman_measures(): provides a simple access to several (quantum) measures in order to characterize quantum states, quantum operators and quantum operations. These measures comprises distances, fidelities, entropies as well as entanglement and non-locality measures of different kind and complexity. The command also supports both, pure and mixed states. Apart from minor changes to the keystrings, the syntax of this command remained rather unchanged when compared with previous versions of the FEYNMAN program [10,18], though it was re-implemented in order to follow the general design principles and to facilitate the further maintenance of the code. Table 5 displays a few selected measures for two- and multi-qubit systems that can be evaluated by this command, and how these measures are divided into subsections. Again, a list of all currently supported keystrings is obtained by:

```
> Feynman_measures();

The following 'keystrings' are supported by this command:
-----------------------------------------------------------
  a) Distances &            affinity                  Bures distance              C-distance
           fidelities::     fidelity                  Fubini-Study distance       Hellinger distance
                            Hilbert-Schmidt distance  HS distance                 trace distance
  b) Entropy measures::     conditional entropy       entropy                     linear entropy
                            mutual information        participation ratio         relative entropy
  c) Entanglement &         concurrence               entanglement of formation   EOF
         non-locality::     entropy of entanglement   I-concurrence               logarithmic negativity
                            negativity
  d) Entanglement,          global entanglement       n-concurrence               n-tangle
        multi-partite::     3-tangle
  e) Quantum operations::   average fidelity         average fidelity 2          C-distance: qoperation
                            J-distance                J-fidelity
                            phase-space contraction   process purity
```

For all further details about this command, we refer the reader to the manual of the FEYNMAN tools. The design (and division of the quantum measures into different subtopics) enables us to *add* further quantum measures in the future as the need for that may arise.

Many of these measures can be applied also in order to describe the time evolution of some property of a quantum registers, if this is analyzed within a particular context or protocol. Owing to the Jamiołkowski isomorphism [32], moreover, there exist a *one-to-one* correspondence between quantum operations and quantum states (of proper dimension). This isomorphism can be utilized to define several quantum measures and separability criteria also for quantum operations which were defined originally for quantum states only.

Table 5
Selected quantum measures that are currently supported by the command Feynman_measures(). A more detailed description of the argument options and how these measures are divided into subsections is given in the manual to the FEYNMAN tools.

Argument option	Explanation
(a) *Distances and fidelities*	
("Bures distance",...)	Bures distance $D_B(\rho,\sigma) = \sqrt{2-2\,\mathrm{Tr}\sqrt{\sqrt{\rho}\sigma\sqrt{\rho}}}$; cf. Ref. [23].
("fidelity",...)	Fidelity $F(\rho,\sigma) = [\mathrm{Tr}(\sqrt{\sqrt{\rho}\sigma\sqrt{\rho}})]^2$; cf. Ref. [24].
("Hilbert-Schmidt distance",...)	Hilbert–Schmidt distance $D_{HS}(\rho,\sigma) = \|\rho-\sigma\|_{HS} = \sqrt{\mathrm{Tr}[(\rho-\sigma)^\dagger(\rho-\sigma)]}$.
("trace distance",...)	Trace distance $D_{Tr}(\rho,\sigma) = \frac{1}{2}\|\rho-\sigma\|_{Tr} = \frac{1}{2}\mathrm{Tr}\sqrt{(\rho-\sigma)^\dagger(\rho-\sigma)}$; cf. Ref. [23].
(b) *Entropy measures*	
("conditional entropy",...)	(Von Neumann) conditional entropy $S(A\|B) = S(\rho_{AB}) - S(\rho_B)$ [25].
("entropy",...)	Von Neumann entropy $S(\rho) = -\mathrm{Tr}(\rho\log_2\rho)$.
("linear entropy",...)	Linearized version of the von Neumann entropy, $S_l(\rho) = \frac{d}{d-1}\left(1-\mathrm{Tr}(\rho^2)\right)$.
("relative entropy",...)	Relative entropy $S(\rho\parallel\sigma) = -S(\rho) - \mathrm{Tr}(\rho\log_2\sigma)$ [26].
(c) *Entanglement and non-locality measures for bipartite systems*	
("concurrence",...)	Concurrence for two qubits, $\mathcal{C}(\rho_{AB}) = \max\{0,\lambda_1-\lambda_2-\lambda_3-\lambda_4\}$, and where the λ_i refer to some eigenvalues in descending order [27].
("entanglement of formation",...)	Entanglement of formation for two qubits, $E_F(\rho_{AB}) = h(\frac{1}{2}+\frac{1}{2}\sqrt{1-\mathcal{C}_{AB}^2})$, where $h(x) = -x\log_2 x-(1-x)\log_2(1-x)$ [27].
("entropy of entanglement",...)	Entropy of entanglement $E_E(\|\psi_{AB}\rangle) = S(\rho_A) = S(\rho_B)$ [28].
("negativity",...)	Negativity $\mathcal{N}(\rho) = \frac{\|\rho^{T_A}\|_{Tr}-1}{d-1}$ [29].
(d) *Entanglement measures for multi-partite systems*	
("global entanglement",...)	Global entanglement $Q(\|\psi\rangle) = 2[1-1/n\sum_{i=1}^n \mathrm{Tr}(\rho_i^2)]$.
("n-concurrence",...)	n-qubit concurrence $C_N(\|\psi\rangle) = 2^{1-(N/2)}\sqrt{2^N-2-\sum_\alpha \mathrm{Tr}\left(\rho_\alpha^2\right)}$ [30].
("n-tangle",...)	A natural generalization of the 3-tangle [31].
("3-tangle",...)	Three-qubit residual entanglement $\tau_{ABC} = \mathcal{C}_{A(BC)}^2 - \mathcal{C}_{AB}^2 - \mathcal{C}_{AC}^2$ with $\mathcal{C}_{A(BC)} = 2\sqrt{\det\rho_A}$
(e) *Measures for quantum operators and channels*	
("C-distance: qoperation",...)	To compute the C-distance $C(\mathcal{E},\mathcal{F}) = C(\rho_\mathcal{E},\rho_\mathcal{F})$ between the two quantum operations $\mathcal{E}$ and $\mathcal{F}$. The C-distance is defined for the (Jamiołkowski) dual states $\rho_\mathcal{E}$ and $\rho_\mathcal{F}$, and is therefore a metric.
("J-fidelity",...)	To compute the Jamiołkowski fidelity $F_J(\mathcal{E},\mathcal{F}) = F(\rho_\mathcal{E},\rho_\mathcal{F})$, and where $\rho_\mathcal{E}$, $\rho_\mathcal{F}$ denote the dual states of $\mathcal{E}$ and $\mathcal{F}$ due to the Jamiołkowski isomorphism [23,32].
("process purity",...)	To compute the process purity $\mathrm{Tr}(\rho_\mathcal{E}^2)$ of a given quantum operation, and where $\rho_\mathcal{E}$ is the dual state of $\mathcal{E}$ as defined by the Jamiołkowski isomorphism [23,32].

Feynman_plot(): This is the central command to display, for instance, the amplitudes of a given quantum state, including mixed states, or the process matrices of some quantum operation in a neat format. The currently supported keystrings are:

```
> Feynman_plot();

The following 'keystrings' are supported by this command:
-----------------------------------------------------------
  a) States & density matrices::   Bloch vector       state                  state: Re
                                   state: Im          probability
  b) Quantum tomography::          process matrix     process matrix: Re     process matrix: Im
```

A plot is usually returned. The FEYNMAN tools automatically select either a 2- or 3-dimensional plot as appropriate for the given input data.

Feynman_qoperation(): This command provides quick access to many predefined quantum operations as they occur in the literature; the presently supported keystrings are:

```
> Feynman_qoperation();

The following 'keystrings' are supported by this command:
-----------------------------------------------------------
  a) Single-qubit::           Pauli                                depolarize
                              amplitude damping                    generalized amplitude damping
                              GAD                                  phase damping
                              phase flip                           bit flip
                              bit+phase flip                       user
  b) Distr. single-qubit::    Pauli: n                             depolarize: n
                              amplitude damping: n                 generalized amplitude damping: n
                              GAD: n                               phase damping: n
                              phase flip: n                        bit flip: n
                              bit+phase flip: n                    user: n
  c) k-qubit::                individual amplitude damping         individual phase flip
                              fully correlated amplitude damping   fully correlated phase flip
```

Until now, we distinguish between single-qubit, distributed single-qubit and k-qubit quantum operations. This command defines (and returns) a quantum operation, $\mathcal{E}(\rho) = \sum_i E_i\,\rho\,E_i^\dagger$, in terms of its operation elements E_i, also known as Kraus operators. For the k-qubit quantum operations, Table 6 displays the currently implemented argument options and the corresponding operation elements.

Table 6
The k-qubit quantum operations that are supported by the command Feynman_ qoperation().

Argument option	List of the returned operation elements
`("individual amplitude damping", p, k)`	Individual amplitude-damping operation that only acts upon one of k qubits at a time. The Kraus operators E_i have a simple tensorial structure $E_i = \sqrt{p/k}\,\sigma_+^{(i)} = \sqrt{p/k}\, I \otimes \cdots \otimes \sigma_+ \otimes \cdots \otimes I$ for $i = 1..k$, and $E_0 = \sqrt{I - \sum_{i=1}^{k} E_i^\dagger E_i}$.
`("individual phase flip", p, k)`	Individual phase-flip operation that only acts upon one of k qubits at a time; it has the Kraus operators $E_i = \sqrt{p/k}\,\sigma_z^{(i)} = \sqrt{p/k}\, I \otimes \cdots \otimes \sigma_z \otimes \cdots \otimes I$ for $i = 1..k$, and $E_0 = \sqrt{I - \sum_{i=1}^{k} E_i^\dagger E_i}$.
`("fully correlated amplitude damping", p, k)`	Correlated amplitude damping operation that acts upon all k qubits at the same time; it has the two Kraus operators $E_0 = \sqrt{I - E_1^\dagger E_1}$ and $E_1 = \sqrt{p}\,\sigma_+^{\otimes k}$.
`("fully correlated phase flip", p, k)`	Correlated phase flip operation that acts upon all k qubits at the same time; it has the two Kraus operators $E_0 = \sqrt{1-p}\, I^{\otimes k}$ and $E_1 = \sqrt{p}\,\sigma_z^{\otimes k}$.

Feynman_qoperator(): Provides quick access to a large set of predefined quantum operators from the literature. The presently supported keystrings are:

```
> Feynman_qoperator();

The following 'keystrings' are supported by this command:
-----------------------------------------------------------
  a) Single-qubit::        I               not               sigma_x              X
                           sigma_y         Y                 sigma_z              Z
                           sigma[-]        sigma[+]          phase                A
                           Hadamard        H                 Rx                   Ry
                           Rz              S                 T                    Euler
                           not^1/2
  b) Two-qubit::           cn              cnot              controlled-phase     controlled-s
                           cs              controlled-z      cz                   swap
  c) Three-qubit::         ccn             Toffoli           Fredkin
  d) k-qubit::             XYZ string      XYZ               Fourier              projector
                           permute         c..c              oracle               Grover step
  e) Distr. single-qubit:: I: n            not: n            sigma_x: n           X: n
                           sigma_y: n      Y: n              sigma_z: n           Z: n
                           sigma[-]: n     sigma[+]: n       phase: n             A: n
                           Hadamard: n     H: n              Rx: n                Ry: n
                           Rz: n           S: n              T: n                 Euler: n
                           not^1/2: n      single: n
  f) Distr. two-qubit::    cn: n           cnot: n           controlled-phase: n  controlled-s: n
                           cs: n           controlled-z: n   cz: n                swap: n
                           two: n
  g) Distr. three-qubit::  ccn: n          Toffoli: n        Fredkin: n           three: n
  h) Distr. k-qubit::      XYZ string: n   XYZ: n           Fourier: n           projector: n
                           permute: n      c..c: n           oracle: n            Grover step: n
                           k-qubit: n
```

From these quantum operators, I here display only those keystrings and arguments for illustration which help obtain the various single-qubit (Table 7) and distributed two-qubit gates (Table 8). The so-called 'distributed' gates are generated in such a way that a given (k-qubit) quantum operator U is applied to a selected set of k qubits, $[i_1, \ldots, i_k]$, but which belong to a n-qubit quantum register, i.e. $1 \le i_k \le n$ for all n. Since, in general, it is not possible then to express the (overall) n-qubit operator as a simple Kronecker product of the k-qubit operator (matrix) and $(n-k)$ identity operators I_{2x2}, the k-qubit operator need to be *distributed* properly over the $2^n \times 2^n$-matrix. This can be achieved, for example, by a permutation of the qubit indices.

In addition to the pre-defined operators from above, a user-defined $2^k \times 2^k$ matrix U may be given explicitly, following the keystring `"user"`, and can be distributed in an analogous way over the n-qubits in order to generate the correct $2^n \times 2^n$ quantum operator.

To analyze the properties of quantum operations (channels) and operators, the commands Feynman_is() and Feynman_measures() can be utilized and provide a quick and reliable access to this information.

3.4. Selected technical features

While the major part of the code remains hidden to the user owing to the library structure of the FEYNMAN tools (as long as the source code is not modified explicitly), there are a few technical features of interest for the daily work with this toolbox. These features enable the user to accelerate the execution or to facilitate the handling of the FEYNMAN tools through the use of type definitions, global variables, or the explicit conversion of data into some different format. Here, I shall give only a brief discussion of such technical issues and refer the reader for further details to the manual.

Type definitions: In order to simplify the implementation and work with the FEYNMAN tools some additional object types are defined internally which can be used in a similar way as the predefined ones. For example, `type(a,qregister)` returns `true` if the variable `a` is a qregister() and `false` otherwise. Analogue type definitions are associated with all data structures in Table 2 [cf. Section 3.2]. These type definitions can be used also to *test* for the type of the parameters that are given to a procedure.

Table 7
Single-qubit quantum gates that are supported by the command Feynman_qoperator().

Argument option	To return the matrix representation of the
("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[+]")	$\sigma_+ = (\sigma_x + i\sigma_y)/2$ (a non-unitary operator !)
("sigma[-]")	$\sigma_- = (\sigma_x - i\sigma_y)/2$ (a non-unitary operator !)
("phase", ϕ)	relative phase shift by $\begin{pmatrix} 1 & 0 \\ 0 & e^{i\phi} \end{pmatrix}$
("A", ϕ)	alternative definition of the phase gate $A = \begin{pmatrix} e^{i\phi} & 0 \\ 0 & e^{-i\phi} \end{pmatrix}$
("Hadamard") or ("H")	H
("Euler",[α, β, γ, δ])	$U = e^{i\alpha} R_z(\beta) R_y(\gamma) R_z(\delta)$
("Rx", θ)	$R_x(\theta)$ rotation
("Ry", θ)	$R_y(\theta)$ rotation
("Rz", θ)	$R_z(\theta)$ rotation
("S")	$S = \begin{pmatrix} 1 & 0 \\ 0 & I \end{pmatrix}$
("T")	$T = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\pi/4} \end{pmatrix}$
("not^1/2")	$\sqrt{\text{NOT}}$
("...", adjoint)	the adjoint (matrix) of the selected gate operation.

Table 8
Distributed two-qubit quantum gates that are supported by the command Feynman_qoperator().

Argument option	To return the matrix representation of the
("cn: n", [m_1, m_2], n) or ("cnot: n", [m_1, m_2], n)	controlled-not operator that acts upon the qubits m_1 and m_2 within an n-qubit gate.
("controlled-phase: n", ϕ, [m_1, m_2], n)	controlled-phase operator that acts upon the qubits m_1 and m_2.
("controlled-s: n", [m_1, m_2], n) or ("cs: n", [m_1, m_2], n)	controlled-phase or *cs* operator that acts upon the qubits m_1 and m_2.
("controlled-z: n", [m_1, m_2], n) or ("cz: n", [m_1, m_2], n)	controlled-z or *cz* operator that acts upon the qubits m_1 and m_2.
("swap: n", [m_1, m_2], n)	swap operator that acts upon the qubits m_1 and m_2.
("two: n", $U_{4\times4}$, [m_1, m_2], n)	distributed operator $U_{4\times4}$ that acts upon the qubits m_1 and m_2.
("...", [m_1, m_2], n, adjoint)	the adjoint (matrix) of the selected operation.

Global variables: Two global variables are (re-) defined in the FEYNMAN tools and initialized at the beginning of each session to the values:

- `Digits := trunc(evalhf(Digits));` this sets the `Digits` variable from its *default* to the maximum value that is supported by the floating point evaluation of the local hardware.
- `Feynman_precision := 10^(-trunc(0.8*Digits));` this defines a numerical noise threshold which is tolerated, for example, when the equality of two objects is tested for or in order to find some quantity negligible. The latter is often done for a sufficiently small imaginary part of a measure that is expected to be real-valued. It is recommended to use the default value above. The `Feynman_precision` variable currently affects the commands Feynman_compute(), Feynman_decompose(), Feynman_is() and Feynman_measures(), respectively.

Properties of quantum operators: A `FAIL` is typically returned if no unique decision can be made about the given property of a quantum operator or some other entity. This is the case, for instance, if no equivalence can be shown due to some (unsuccessful) simplification. Since some numerical noise is typically accepted by the command Feynman_is() owing to the global variable `Feynman_precision`, a call to this procedure may return `true` for some (floating-number) operator where MAPLE's internal procedure would return `false`.

Matrix manipulations: A few such manipulations, such as to simplify or to take the square root of a matrix, are supported by the command Feynman_compute(), section (i). These (additional) features help circumvent some minor problems with MAPLE's internal representation of matrices. A matrix is (typically) returned. The global variable `Feynman_precision` is used here but can be overwritten by some explicitly given threshold in this case.

Further details about the implementation of the FEYNMAN tools can be found in the manual to the code and in numerous in-line comments.

3.5. Further extensions

The design of the FEYNMAN tools is flexible enough to include a number of extensions without that the structure or the number of the commands need to be enlarged (much). For example, first steps have been undertaken already to incorporate various tasks from quantum state or process tomography into the command Feynman_perform(), cf. Table 3. The same command might be utilized also to perform (so-called) *twirl* operations upon some given quantum state. Indeed, many computational tasks as summarized in Table 1 can be readily incorporated into the FEYNMAN tools. With the present re-design, we therefore hope to improve the conceptual and computational strength of the FEYNMAN tools and to prepare it for a long life-cycle.

3.6. Installation and distribution

As previously [11], the FEYNMAN program is organized as a module (package) for MAPLE and currently contains about 150 (sub-) procedures in some hierarchical order. From these procedures, only the auxiliary procedures from Table 2 and the main commands from Table 3 are *seen* by the user and may help in the interactive work. Of course, the (main) commands can be used also as *building blocks* to adapt or extend the program towards more advanced applications.

The FEYNMAN tools are distributed as a compressed archive `Feynman-2013.tgz`; this archive file comprises the `Feynman.ind` and `Feynman.lib` library files in the subdirectory `lib`, the source code `Feynman-source-2013.txt` as well as the manual `Feynman-manual-2013.pdf`. Moreover, this archive also includes the MAPLE worksheets `Feynman-commands-all-2013.mw` and `Feynman-commands-measures-2013.mw` which contain small examples for (almost) all implemented functions and argument options. By using a copy of these files, therefore, one can easily check the local implementation and compare the output from these examples with the given data. In addition, the worksheet `Feynman-examples-2013.mw` contain all examples from Section 4, together with some comments and guidance on the use of the program, as well as `Feynman-examples-2013-source` to provide the source of three procedures for the example 4.3. Finally, a short `Read.me` explains the installation of the library; after the proper installation of the code, the FEYNMAN tools can be loaded like any other module of MAPLE by using the command `with(Feynman)`.

4. Examples

To illustrate the interactive work with the FEYNMAN tools, let us discuss three examples here. Apart from the simple characterization of some single-qubit channels, these examples include the (simulation of) quantum measurements on some given state as well as a brief analysis of the entanglement swapping between two parties, but with a *delay* in doing the Bell-state measurements. In this section, we assume that the FEYNMAN tools has been 'loaded' into the current session by using the `with(Feynman)` command and that the MAPLE prompt is available for interactive work. Moreover, a colon is often used below (instead of a semicolon at the end of the input lines) in order to suppress the rather large Maple output that is printed to screen. For the sake of convenience, these examples are distributed also by the worksheet `Feynman-examples-2013.mws` together with the code.

4.1. Characterization of single-qubit channels: entanglement vs. purity

Following Ref. [11], let us first analyze the properties of some given quantum channels (quantum operations). For the sake of simplicity, we here consider two well-known (single-qubit) channels, namely, the *bit-flip* and *amplitude-damping* channels. Within the FEYNMAN tools, these channels can be defined and assigned to some variable either in a compact (keystring) notation

```
> assume(p::real, p>0, p<1);
> channel_bf := qoperation("bit flip",p);
                            channel_bf := Feynman:-qoperation("bit flip", p~)

> channel_ad := qoperation("amplitude damping", p);
                         channel_ad := Feynman:-qoperation("amplitude damping", p~)
```

or explicitly in terms of their operation elements

```
> c_bf := Feynman_qoperation("bit flip",p);
                                  [        1/2              ]  [           1/2]
                                  [(1 - p~)            0    ]  [ 0      p~   ]
                       c_bf := [[                           ], [               ]]
                                  [                      1/2]  [ 1/2           ]
                                  [     0         (1 - p~)   ]  [p~          0  ]

> c_ad := Feynman_qoperation("amplitude damping", p);
                                    [1         0     ]  [       1/2]
                             c_ad := [[               ], [0    p~   ]]
                                    [             1/2]  [           ]
                                    [0    (1 - p~)   ]  [0      0  ]
```

Both representations are handled rather equally within the FEYNMAN tools, although some commands request a particular format in order to proceed properly. While the bit-flip channel is known to be *unital*, i.e. it maps the identity operator I upon itself, the amplitude damping describes an non-unital, e.g. dissipative, process. In the FEYNMAN tools, we can easily test any given channel for this property

```
> Feynman_is("unital", c_bf), Feynman_is("unital", c_ad);
                                                  [1    0]
                                "Sum_i E_i^+ I E_i = ", [      ]
                                                  [0    1]

                                                  [1 + p~      0   ]
                                "Sum_i E_i^+ I E_i = ", [                ]
                                                  [  0       1 - p~]

                                          true, FAIL
```

Indeed, the amplitude-damping channel is dissipative for any *non-zero* value of p, for instance, $p = 0.1$

```
> Feynman_is("unital", subs(p=0.1, c_ad));
                                          [1.100000000          0.     ]
                          "Sum_i E_i^+ I E_i = ", [                      ]
                                          [    0.          0.9000000001]

                                          false
```

In addition, we can test these channels also for a number of other properties

```
> Feynman_is("completely positive", c_bf),  Feynman_is("completely positive", c_ad);
                                              true, true

> Feynman_is("trace preserving", c_bf),  Feynman_is("trace preserving", c_ad);
                                              true, FAIL

> Feynman_is("hermiticity-preserving", c_bf),  Feynman_is("hermiticity-preserving", c_ad);
                                              true, true
```

and where the FAIL arises here again for the same reasons as above (that p need to be specified explicitly). To re-call some details about these properties, a quantum operation is said to be *completely positive* if the corresponding dual state is positive, and it preserves the *hermiticity* if the quantum operation maps hermitian matrices upon hermitian matrices. The latter property is equivalent also to the fact that the *dual* state of the quantum operation is hermitian. Of course, this *one-to-one* correspondence between quantum operations and (dual) states due to the Jamiołkowski isomorphism can be utilized also in order to explore further properties for quantum channels, such as the entanglement and purity [33]. In the FEYNMAN tools, the dual state is easily generated by

```
> dual_bf := Feynman_evaluate("qoperation: dual state", channel_bf):
> dual_ad := Feynman_evaluate("qoperation: dual state", channel_ad):
```

but where we omit here to print the result explicitly.

From these dual states, we can evaluate for example the concurrence of each channel as a measure of entanglement as well as the (so-called) participation ratio as the inverse of the purity, $\mathrm{Tr}(\rho^2)$. In the FEYNMAN tools, this is done by calling the command Feynman_measures() with some proper keystrings

```
> concur_bf := evalf( Feynman_measures("concurrence", dual_bf) );
                                 concur_bf := 2. max(p~, 1. - 1. p~) - 1.

> concur_ad := evalf( Feynman_measures("concurrence", dual_ad) );
                                                          1/2
                                      concur_ad := (1. - 1. p~)

> purity_bf := evalf( 1/Feynman_measures("participation ratio", dual_bf) );
                                                                  2     2
                         purity_bf := 4. (0.5000000000 - 0.5000000000 p~)  + p~

> purity_ad := evalf( 1/Feynman_measures("participation ratio", dual_ad) );
                                                            2                                  2
     purity_ad := 0.7500000000 - 0.5000000000 p~ + 0.2500000000 p~  + (0.5000000000 - 0.5000000000 p~)
```

Using these expressions, we can readily display the 'entanglement-vs.-purity' diagram; to compare the graphs for both channels within a single figure, we first define two independent plots and display them afterwards by the MAPLE command `plots[display]()`,

```
> plot_bf := plot( [purity_bf, concur_bf, p=0..1], Purity=0..1, Concurrence=0..1):
> plot_ad := plot( [purity_ad, concur_ad, p=0..1]):
> plots[display]( [plot_bf, plot_ad] );
```

This plot is shown in Fig. 1 and demonstrate that the dual states are completely entangled for $p = 0$ but that this entanglement get lost differently as the purity decreases or the 'coupling' to the environment is enhanced until no entanglement is left for $p = 1$. Of course, similar plots can be drawn quite easily for other channels and might help explore the effect of noise upon different (classes of) quantum states.

4.2. Measurements on quantum states

In many protocols in QIP, measurements on quantum states and quantum registers need to be performed quite frequently, either on single qubits, some particular subsystem or with regard to all parties of the given system. To demonstrate how such quantum measurements can be simulated within the FEYNMAN tools, let us consider the 3-qubit GHZ state

$$|\psi_{\mathrm{GHZ}}\rangle = \frac{1}{\sqrt{2}}\,(|000\rangle + |111\rangle) \tag{1}$$

and apply a 3-qubit Fourier transform upon this state, $|\psi_{\mathrm{F3}}\rangle := F^{\mathrm{Fourier}}\,|\psi_{\mathrm{GHZ}}\rangle$. One may then ask, for instance, for the probabilities p_+ and p_- that, after a projective measurement, the system is found in the states $|+++\rangle$ and $|---\rangle$, respectively, and where as usual $|\pm\rangle = \frac{1}{\sqrt{2}}\,(|0\rangle \pm |1\rangle)$. To perform this – simple but rather tedious – computation, the FEYNMAN tools support different computational paths, in dependence of how this (projective) measurement is tackled or interpreted. To start with, let us first evaluate the Fourier transform of a 3-qubit GHZ state,

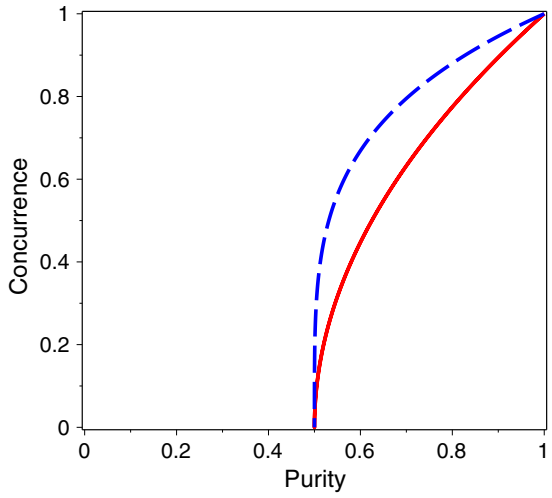

Fig. 1. 'Entanglement-vs.-purity' diagram for the bit-flip (red, solid line) and the amplitude-damping channel (blue, dashed line).

```
> ghz := Feynman_define("GHZ", 3):   Feynman_print(ghz);
                                        1/2          1/2
                                       2    |000>   2    |111>
                                       ---------- + ----------
                                           2            2

> wt  := qoperator("Fourier",3);
                                wt := Feynman:-qoperator("Fourier", 3)

> wa  := Feynman_apply(wt,ghz);
                                                [          1/2          ]
                                                [                       ]
                                                [                      1/2]
                                                [1/4 + (1/8 - 1/8 I) 2   ]
                                                [                       ]
                                                [      1/4 - 1/4 I      ]
                                                [                       ]
                                                [                      1/2]
                                                [1/4 - (1/8 + 1/8 I) 2   ]
                        wa := Feynman:-qregister(3, [                       ])
                                                [           0           ]
                                                [                       ]
                                                [                      1/2]
                                                [1/4 - (1/8 - 1/8 I) 2   ]
                                                [                       ]
                                                [      1/4 + 1/4 I      ]
                                                [                       ]
                                                [                      1/2]
                                                [1/4 + (1/8 + 1/8 I) 2   ]
```

and also assign the states $|+++\rangle$ and $|---\rangle$ to some proper variables (but without displaying them explicitly here).

```
> w3p := Feynman_define("string", "+++"):   w3m := Feynman_define("string", "---"):
```

Below, we will show four different paths in order to compute the probabilities p_+ and p_-.
(i) Most simply, perhaps, they are obtained from the projections

$$p_+ = |\langle +++|\psi_{F3}\rangle|^2 \tag{2}$$

$$p_- = |\langle ---|\psi_{F3}\rangle|^2 \tag{3}$$

or

```
> pp := abs( Feynman_evaluate("inner product", w3p, wa) )^2;
                                              pp := 1/2

> pm := abs( Feynman_evaluate("inner product", w3m, wa) )^2;   evalf(pm);
                                             / 1/2      \2
                                             |2         |
                                        pm := |---- - 1/4|  + 1/32
                                             \ 8        /

                                             0.03661165237
```

(ii) Alternatively, we can perform a subsequent measurement on all three qubits along the x-axis and multiply the corresponding probabilities to measure the values $+1$ or -1 in each case. While the overall probabilities are independent of the sequence in which the qubits are measured, we here choose the series 1–2–3 to avoid confusion with the qubit indices. After some qubit is projected, of course, we will need to proceed with the corresponding *post-measurement* state. To simulate the measurement for the first qubit, we invoke

```
> wb := Feynman_measure("projective: X", 1, wa, {"+1"});
                                                wb := T
```

and specify already in this call to the procedure that the output is required only for the outcome `"+1"`. Typically, a table `T` is returned by the command Feynman_measure() in which the outcomes, probabilities and post-measurement states are given by the (table) entries `T[outcome]`, `T[probabilities]` and `T[post_ms]`, respectively.

For the x-projection of qubit 1 with given outcome `"+1"`, only one probability and post-measurement state is returned, and they have the values

```
> p1 := op(1, wb[probabilities]);   ws := op(1, wb[pm_states]):
                                                p1 := 1/2
```

Making analogue measurements also on qubits 2 and 3

```
> wb := Feynman_measure("projective: X", 2, ws, {"+1"}):
> p2 := op(1, wb[probabilities]):   ws := op(1, wb[pm_states]):
> wb := Feynman_measure("projective: X", 3, ws, {"+1"}):
> p3 := op(1, wb[probabilities]):
```

we obtain the overall probability

```
> pp := p1 * p2 * p3;
                                                pp := 1/2
```

and similar also for p_-

```
> wb := Feynman_measure("projective: X", 1, wa, {"-1"}):
> p1 := op(1, wb[probabilities]):   ws := op(1, wb[pm_states]):
> wb := Feynman_measure("projective: X", 2, ws, {"-1"}):
> p2 := op(1, wb[probabilities]):   ws := op(1, wb[pm_states]):
> wb := Feynman_measure("projective: X", 3, ws, {"-1"}):
> p3 := op(1, wb[probabilities]):
> pm := simplify(p1 * p2 * p3);  evalf(pm);
                                                    1/2
                                                   2
                                          pm := 1/8 - ----
                                                    16

                                           0.03661165238
```

(iii) Apart from a projective measurement, we can obtain the probabilities p_+ and p_- from a POVM measurement by using the POVM elements $E_1 = |+++\rangle\langle+++|$, $E_2 = |---\rangle\langle---|$ and $E_3 = I - E_1 = E_2$, respectively. With the Feynman tools, such a measurement is simulated by

```
> E1 := Feynman_evaluate("outer product", w3p, w3p):
> E2 := Feynman_evaluate("outer product", w3m, w3m):
> E3 := Feynman_qoperator("XYZ", "III") - E1 - E2:
```

and where the command Feynman_qoperator() has been used in the last line to generate the $I_{8\times8} = I \otimes I \otimes I \equiv III$ unit operator from its keystring notation. Again, the command Feynman_measure() is invoked to perform the POVM measurement

```
> wm := Feynman_measure("POVM", [E1, E2, E3], wa);
                                                wm := T

> wm[outcome], wm[probabilities], evalf(wm[probabilities]);
                                      1/2         1/2
                                     2           2
       ["E1", "E2", "E3"], [1/2, 1/8 - ----, 3/8 + ----], [0.5000000000, 0.03661165238, 0.4633883476]
                                      16          16
```

Of course, no post-measurement states are available for a POVM measurement here.

(iv) Finally, we can determine the probabilities p_+ and p_- from a generalized measurement with a proper set of measurement operators, M_i, $i = 1..8$, as given, for instance, by the projectors of the 3-qubit 'diagonal' basis. These projectors are easily generated by

```
> wc := Feynman_qbasis("diagonal",3):
> wd := []:
> for  i from 1 to nops(wc)  do
>    wd := [ op(wd), Feynman_evaluate("outer product", wc[i], wc[i]) ];
> end do:
```

and correspond as first and last element (of the list `wc` of the 3-qubit diagonal states) to $|+++\rangle$ and $|---\rangle$, respectively. This can be seen also from the 'overlap' of the states `w3p` and `w3m` with these list elements of `wc`,

```
> Feynman_evaluate("inner product", op(2,w3p), wc[1]);
                                                1

> Feynman_evaluate("inner product", op(2,w3m), wc[8]);
                                                1
```

To perform the requested *generalized* measurement, we invoke

```
> wn := Feynman_measure("generalized", wd, wa);
                                                wn := T

> wn[outcome]; wn[probabilities]; evalf(wn[probabilities]);
                          ["M 1", "M 2", "M 3", "M 4", "M 5", "M 6", "M 7", "M 8"]

                                                1/2         1/2         1/2         1/2
                                               2           2           2           2
                        [1/2, 0, 0, 0, 1/2 + ----, 1/2 - ----, 1/2 + ----, 1/2 - ----]
                                                16          16          16          16

              [0.5000000000, 0., 0., 0., 0.2133883476, 0.03661165238, 0.2133883476, 0.03661165238]
```

and easily read-off the desired result for p_+ and p_- from the list of probabilities.

This second example shows that the FEYNMAN tools support quite different (computational) techniques for deriving some requested result, a flexibility that is often very desirable for tackling complex tasks.

4.3. Delayed-choice entanglement swapping

Entanglement swapping is a well-known quantum phenomenon that has been shown and explored for different qubit systems [34,35]. Most easily, it is explained for two pairs of (initially) entangled qubits, e.g., photons. From each of these pairs, one qubit is given to, say, Charlie, while the other (two) are sent to Alice and Bob typically. If Charlie projects his two qubits upon some entangled state, i.e. if he performs a Bell-state measurement, the other two qubits become entangled although they have never interacted with each other. This surprising 'swap' of quantum correlations is often regarded as a direct consequence of the stochastical outcome of (quantum) measurements but can be also observed, even if Charlie makes his projection upon an entangled state *after* that Alice and Bob have measured already the (polarization) state of their qubits. This gedanken experiment is known as 'delayed-choice entanglement swapping'; it was originally discussed by Peres [36] and has been demonstrated recently with four photons by Ma et al. [35].

We can make use of the FEYNMAN tools to simulate the entanglement swapping, either for a *prior* or *delayed* choice that Charlie projects his two qubits upon an entangled state. To simplify the notations (and the MAPLE output) below, we shall start from the product of the Bell states $\psi_{12} = \Phi^+$ and $\psi_{34} = \Psi^+$,

$$\psi = \psi_{12} \otimes \psi_{34}, \tag{4}$$

and assume that qubits 1 and 3 belong to Charlie, while Alice receives the qubit 2 and Bob qubit 4, respectively. Because of the symmetry of the Bell states, however, any other distribution of the qubits would work similarly well as long as Charlie receives one qubit from each Bell state above. Moreover, we also assume that Alice and Bob project their qubits upon the *x*-axis (for measuring the polarization), although any other *common* choice would give rise to a similar measurement statistics as discussed below.

The peculiar behavior of quantum systems in the entanglement swapping procedure, and especially for a *delayed* choice of Charlie's projection, can be easier understood if we begin with the *standard* protocol. In this case, we need only to show that, after the projection of qubits 1 and 3 upon the Bell states, the remaining two qubits of Alice and Bob are fully entangled with each other. To demonstrate this by means of the FEYNMAN tools, let us assign the two Bell states and the product state (4) to the variables

```
> psi_12 := Feynman_define("Bell: Phi+"):   psi_34 := Feynman_define("Bell: Psi+"):
> psi := Feynman_evaluate("Kronecker product", psi_12, psi_34):
```

A Bell-state measurement on the qubits 1 and 3 is performed by

```
> wm := Feynman_measure("projective: Bell", [1,3], psi):
> wm[outcome]; wm[probabilities];
                                    ["Phi+", "Phi-", "Psi+", "Psi-"]

                                          [1/4, 1/4, 1/4, 1/4]

> psi_24 := wm[pm_states]:
```

and provides us immediately with the possible outcomes and probability distribution of such a measurement as well as the corresponding *post-measurement* states (which we have assigned separately to the variable `psi_24` for later use). As expected, all four probabilities for the different Bell states are the same. While we here omit the explicit printout of the post-measurement states, we can easily demonstrate that they are fully entangled by calculating the concurrence of these (two-qubit) states, and after Charlie's qubits (1, 3) are traced out:

```
> for  i  from 1 to 4 do
>    wa := Feynman_compute("partial trace", psi_24[i], [1,3]):
>    wb := Feynman_measures("concurrence", wa):
>    lprint("Concurrence for post-meausurement state ", i, " is ", wb):
> end do:
"Concurrence for post-meausurement state ", 1, " is ", 1
"Concurrence for post-meausurement state ", 2, " is ", 1
"Concurrence for post-meausurement state ", 3, " is ", 1
"Concurrence for post-meausurement state ", 4, " is ", 1
```

Of course, the same result would be obtained if two other Bell states were applied in order to construct the product state in Eq. (4) above.

```
#
Charlies_prior_choice := proc(keystring, psi, N)
   #
   # This procedure performs  N projective measurements on qubits 1 and 3, either upon the Bell states
   # or the z-axis, and ‘‘before’’ the x-projection is made on qubits 2 (Alice) and 4 (Bob).
   #                      ------
   ...
   #
   for  i  from 1 to N  do
      if    keystring = "Bell: Phi+"  then
         #  ------------------------
         wm := Feynman_measure("projective: Bell", [1,3], psi, {"random"}):
         if  op(1, wm[outcome]) = "Phi+"  then
            ws := op(1, wm[pm_states]);
         else
            next;
         end if;
      elif  keystring = "z-axis: ++"  then
         #  ------------------------
         wm := Feynman_measure("projective: Z", 1, psi, {"random"}):
         wm_out := op(1, wm[outcome]);   wm_pms := op(1, wm[pm_states]);
         wc := Feynman_measure("projective: Z", 3, wm_pms, {"random"}):
         wc_out := op(1, wc[outcome]);
         if  wm_out = 1  and  wc_out = 1  then
            ws := op(1, wc[pm_states]);
         else
            next;
         end if;
      end if;
      #
      wa     := Feynman_measure("projective: X", 2, ws, {"random"}):
      wa_out := op(1, wa[outcome]);   wa_pms := op(1, wa[pm_states]);
      wb     := Feynman_measure("projective: X", 4, wa_pms, {"random"}):
      wb_out := op(1, wb[outcome]);
      #
      if    wa_out = 1  and  wb_out = 1  then  wh[1] := wh[1] + 1;
      elif  wa_out = 1  and  wb_out =-1  then  wh[2] := wh[2] + 1;
      elif  wa_out =-1  and  wb_out = 1  then  wh[3] := wh[3] + 1;
      elif  wa_out =-1  and  wb_out =-1  then  wh[4] := wh[4] + 1;
      end if;
   end do;
   #
   listplot(wh, color="Red", ...);
end proc:
```

Fig. 2. MAPLE source code of the procedure Charlies_prior_choice() as invoked in example 4.3. Only the essential part is shown here, while the complete source of this routine is provided in the file `Feynman-examples-2013-source` together with the code. See text for further explanations.

The four concurrencies for the post-measurement states above clearly show that any subsequent measurement of qubits 2 and 4 (for example, for their projection upon the x-axis) give always rise to a complete correlation in the corresponding measurement statistics. In particular, we expect the outcomes $++$ and $--$ with equal probability for the two Bell states Φ^+ and Φ^-, and the outcomes $+-$ and $-+$ for Ψ^+ and Ψ^-, respectively. Indeed, this is easily confirmed by means of the FEYNMAN tools, if we simulate the outcome of some repeated measurement by a call to the (short) procedure

```
> Charlies_prior_choice("Bell: Phi+", psi, 200);
```

and for which the source code is displayed in Fig. 2. In this procedure, we perform N Bell-state measurement on qubits 1 and 3 but collect the outcomes from the x-projection of qubits 2 and 4 (of Alice and Bob) only if Charlie found before the state `"Phi+"`. In the procedure Charlies_prior_choice(), Charlie's outcome and the corresponding post-measurement state is chosen randomly due to the calculated probabilities. For a call of Charlies_prior_choice() with $N = 200$, a histogram of the outcomes for a projection of qubits 2 and 4 are displayed in Fig. 3(left panel); it clearly shows the 'correlation' between the two qubits by obtaining (with roughly the same probability) the two outcomes $++$ and $--$ but never $+-$ or $-+$ in this selected case. Similar 'correlated' histograms would be found if another outcome is chosen for Charlie's Bell-state measurement or if Alice' and Bob's qubits (2 and 4) were projected upon some other axis.

The complete entanglement (quantum correlations) of the qubits 2 and 4 is of course in contrast to what we expect, if Charlie projects his qubits 1 and 3 separately upon any axis, for instance, the z-axis. Such a set-up of the measurement will project Alice' and Bob's qubits upon some separable state, and will lead to equal probabilities for the four outcomes $++$, $+-$, $-+$ and $--$, owing to the symmetry of the Bell states. We can simulate such a measurement statistics by the same procedure

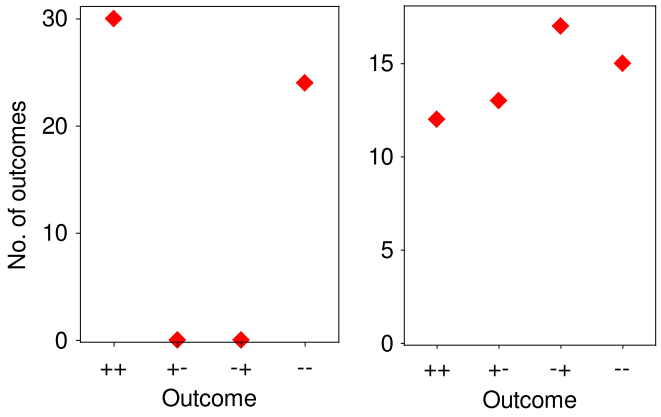

Fig. 3. Measurement statistics for the qubits 2 (Alice) and 4 (Bob), if Charlie has projected his qubits already *before*: for a projection upon the Bell states and for the outcome Φ^+ (left panel) as well as for the projection upon the z-axis and the outcome $++$ (right panel).

```
> Charlies_prior_choice("z-axis: ++", psi, 200);
```

but by making the projection of qubits 1 and 3 upon the z-axis and by choosing the outcome $++$. In the procedure Charlies_prior_choice(), this set-up and selected outcome is chosen by the keystring `"z-axis: ++"`; cf. Fig. 2. For this latter call of Charlies_prior_choice() with $N = 200$, a histogram is shown in Fig. 3(right panel). Indeed, a very similar statistics is obtained for all possible projections $++$, $+-$, $-+$ and $--$ of qubits 1 and 3, in line with the separability of Alice' and Bob's qubits. Fig. 3 therefore shows that entanglement swapping requires the projection upon some entangled state, and *before* the qubits 2 and 4 are measured separately.

In Peres' gedanken experiment [36], and in contrast to the standard procedure of entanglement swapping, Charlie decides independently (every time) of whether he projects his qubits 1 and 3 upon some Bell state, or separately upon any axis (the z-axis in our example below). Peres then raised the question of how Alice' and Bob's qubits are correlated with each other if they perform their measurement *before* Charlie has made his choice. As discussed in Ref. [36] and nicely demonstrated in a recent experiment [35], the answer to this question depends on Charlie's particular choice of projection, and for which outcome he collects the measurement statistics of Alice and Bob. In our example, the entanglement swapping with such a *delayed* choice of Charlie's measurement is simulated by the procedure Charlies_delayed_choice() as listed in Fig. 4. This procedure collects the outcome of Alice' and Bob's measurements for three different scenarios in which Charlie *delays* his measurement on qubits 1 and 3: (i) He projects them upon the z-axis and collects the measurement statistics of Alice and Bob for the outcome $++$; (ii) he projects his qubits upon the Bell states but with no further selection of a particular outcome; and (iii) Charlie projects his qubits upon the Bell states and collects the measurement statistics if he found his qubits in the Φ^+ state. Using the procedure Charlies_delayed_choice(), these scenarios are realized by

```
> Charlies_delayed_choice("z-axis: ++", psi, 200);
> Charlies_delayed_choice("Bell: all",  psi, 200);
> Charlies_delayed_choice("Bell: Phi+", psi, 200);
```

and the (three) corresponding histograms for the measurements of the qubits 2 and 4 along the x-axis are shown in Fig. 5. A swap of entanglement (definite correlation) upon Alice and Bob's qubits is obtained even though the Bell state measurement is made *afterwards*, if the measurement statistics is collected for a well defined outcome of Charlie's measurement.

5. Summary and outlook

The design and implementation of the FEYNMAN tools has been presented in this work together with a short account on its (potential) application in quantum information science. To facilitate their use and maintenance, these tools have been re-designed with the goal to establish a high-level (computer) language that is capable to deal with the physics of finite n-qubit systems, from frequently required tasks to mathematically advanced projects in QIP. In particular, the present implementation supports a large number of computational tasks, including the definition, manipulation and parametrization of quantum states, the evaluation of quantum measures and quantum operations, the evolution of quantum noise in discrete models, quantum measurements and state estimation, and several others.

In order to provide an interactive and user-friendly environment for quantum computations, emphasis was placed on a *syntax* that is close to the mathematical notation and its use in the literature, and which can be generalized to solve computational tasks at even higher degree of complexity. Therefore, the present design is based on a small but rather powerful set of *keystring*-driven commands that is kept as flexible as possible to be combined with other languages and computational environments. With the present implementation and extension, we therefore hope to make these tools more versatile for the analysis of quantum registers and the simulation of QIP protocols. Apart from daily research, the tools may help also in following derivations in the literature and for teaching the basic elements of quantum information theory.

The FEYNMAN tools are designed as an *open* environment within MAPLE in order to support both, symbolic *and/or* numerical computations. This enables us and other users to extend these tools into new directions. Several such new functions and building blocks were outlined in Section 2.1, and first steps towards their realization have been undertaken already. This open design might stimulate also plans to combine the FEYNMAN tools with other software projects with, for instance, more emphasis on numerical computations. Owing to the large number of possible (and surely desirable) extensions of these and additional packages, collaboration with other groups will be appreciated.

```
Charlies_delayed_choice := proc(keystring, psi, N)
   #
   # This procedure performs  N projective measurements on qubits 1 and 3, either upon the Bell states
   # or the z-axis, and ‘‘after’’ that the x-projection is made on qubits 2 (Alice) and 4 (Bob).
   #
   ...
   #
   for  i  from 1 to N  do
      # First make the x-projections of qubits 2 and 4
      wa     := Feynman_measure("projective: X", 2, psi, {"random"}):
      wa_out := op(1, wa[outcome]);   wa_pms := op(1, wa[pm_states]);
      wb     := Feynman_measure("projective: X", 4, wa_pms, {"random"}):
      wb_out := op(1, wb[outcome]);   wb_pms := op(1, wb[pm_states]);
      #
      if    keystring = "Bell: Phi+"  then
         #  ------------------------
         wm := Feynman_measure("projective: Bell", [1,3], wb_pms, {"random"}):
         if  op(1, wm[outcome]) <> "Phi+"  then
            next;
         end if;
      elif  keystring = "Bell: all"  then
         #  ------------------------
         wm := Feynman_measure("projective: Bell", [1,3], wb_pms, {"random"}):
      elif  keystring = "z-axis: ++"  then
         #  ------------------------
         wm := Feynman_measure("projective: Z", 1, wb_pms, {"random"}):
         wm_out := op(1, wm[outcome]);   wm_pms := op(1, wm[pm_states]);
         wc := Feynman_measure("projective: Z", 3, wm_pms, {"random"}):
         wc_out := op(1, wc[outcome]);
         if  wm_out <> 1  or  wc_out <> 1  then
            next;
         end if;
      end if;
      #
      if    wa_out = 1  and  wb_out = 1  then  wh[1] := wh[1] + 1;
      elif  wa_out = 1  and  wb_out =-1  then  wh[2] := wh[2] + 1;
      elif  wa_out =-1  and  wb_out = 1  then  wh[3] := wh[3] + 1;
      elif  wa_out =-1  and  wb_out =-1  then  wh[4] := wh[4] + 1;
      end if;
   end do;
   #
   listplot(wh, color="Red", ...);
end proc:
```

Fig. 4. The same as Fig. 2 but for the procedure Charlies_delayed_choice().

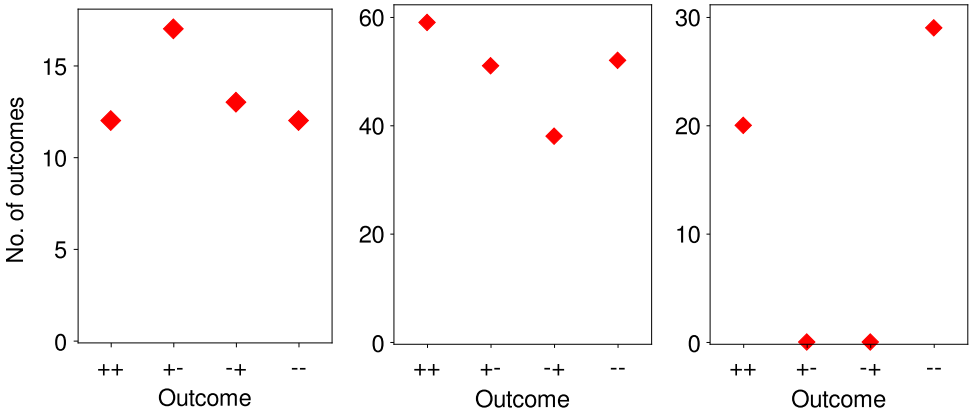

Fig. 5. The same as Fig. 3 but for Charlie's delayed projection of his qubits upon the z-axis and for the outcome ++ (left panel); upon the Bell states and independent of the particular outcome (middle panel); and for a projection upon the Bell states and the outcome Φ^+ (right panel).

Acknowledgment

The work of Thomas Radtke on prior versions of these tools is greatly acknowledged.

Appendix A. Quantum operators and quantum registers

Here, two examples are shown of how the basic data structures are defined and described in the manual to the FEYNMAN tools.

- qoperator()
  Auxiliary procedure to represent (the distributed version of) some pre-defined single- or multi-qubit quantum operator.
  Output: An unevaluated call to qoperator() is returned.
  Argument options:
  * The same parameter lists (argument options) as for Feynman_qoperator() can be used here.
  Additional information:
  * In practice, this procedure provides a data structure to improve the handling and readability of quantum operators within the FEYNMAN tools. An evaluation of the explicit matrix representation of these operators is obtained by means of the main command Feynman_qoperator().
  * Apart from the number (and type) of arguments, no further test is made on them w.r.t. the keystrings or arguments that are supported by Feynman_qoperator().
  See also: Feynman_qoperator(), qoperation().
- qregister(qbit$_1$, qbit$_2$, ..., qbit$_n$)
  Auxiliary procedure to define a n-qubit quantum register in terms of the qubits qbit$_1$(), qbit$_2$(), ..., qbit$_n$(), i.e. as product state $(a_1|0\rangle + b_1|1\rangle) \otimes (a_2|0\rangle + b_2|1\rangle) \otimes \cdots \otimes (a_n|0\rangle + b_n|1\rangle)$.
  Output: A quantum register qregister(n, V) is returned where n is the number of qubits and V a 2^n-dimensional state vector.
  Argument options:(n, V) to represent an n-qubit quantum register where V must be a valid 2^n-dimensional state vector.
  * (n, M_ρ) to represent an n-qubit quantum register where M_ρ must be a valid $2^n \times 2^n$ density matrix $\rho = \sum_{i,j=0}^{2^n-1} \rho_{ij} |i\rangle \langle j|$.
  Additional information:
  * A quantum register can in general contain any number of qubits.
  * This procedure provides a data structure in order to keep together the information about the – pure or mixed – state of a n-qubit quantum system. It also improves the handling and communications within the program.
  See also: qbit(), qoperation(), qoperator().

Appendix B. Definition of *n*-qubit quantum states

Description of the main command Feynman_define() as taken from the manual to the FEYNMAN tools. As typical, the argument options are divided into (sub-) sections in order to facilitate the use and *extension* of these tools by new functions and parameters. Furthermore, some *additional information* and closely related commands are listed here as for most commands.

- Feynman_define()
  Defines various quantum states that occur frequently in the literature or textbook examples.
  Output: A qregister() is typically returned; a list of all presently supported keystrings is printed to screen if called without arguments.
  Argument options:

  (a) Linear combinations and mixtures
  * ("state", c_1*cbs(...) + c_2*cbs(...) + ... + c_n*cbs(...)) to return a qregister() in the given linear combination of the computational basis states (cbs). The returned qregister() contains a state vector but no further test is made on its normalization.
  * ("mixture", p_1*qregister(...) + p_2*qregister(...) + ... + p_n*qregister(...)) to return a qregister() in the mixed state with given probabilities p_i. The returned qregister() contains a (density) matrix but no further test is made on its normalization.

  (b) Diagonal states $|+\rangle$, $|-\rangle$, $|0\rangle$, $|1\rangle$ and products thereof
  * ("string", "+") or ("string", "–") or ("string", "0") or ("string", "1") or ("string", "L") or ("string", "R") in order to return a one-qubit quantum register in the state $|+\rangle = (|0\rangle + |1\rangle)/\sqrt{2}$, $|-\rangle = (|0\rangle - |1\rangle)/\sqrt{2}$, $|0\rangle$, $|1\rangle$, $|L\rangle = (|0\rangle + i|1\rangle)/\sqrt{2}$ or $|R\rangle = (|0\rangle - i|1\rangle)/\sqrt{2}$, respectively.
  * ("string", "+–01...+") to return an n-qubit quantum register in the pure product state $|+, -, 0, 1, \ldots, +\rangle$; any combination (and length) of "+", "–", "0", "1", "L" or "R" is valid here.

  (c) Initialization and random states
  * ("equal", n) or ("Walsh-Hadamard", n) to return a qregister() with n qubits in an equal superposition of all basis states, i.e. in the state $(H|0\rangle)^{\otimes n} = \frac{1}{\sqrt{2^n}} \sum_x |x\rangle$. This is equal to a Hadamard transform on every single qubit of the initial state $|0\rangle^{\otimes n}$.
  * ("random", n) to return a qregister() with n qubits in a random mixed state, i.e. some complex, positive and hermitian trace-normalized density matrix.

  (d) Predefined states from the literature
  * ("Bell: Phi+") or ("Bell: Phi–") or ("Bell: Psi+") or ("Bell: Psi–") to return one of the Bell states $|\Phi^\pm\rangle = \frac{1}{\sqrt{2}}(|00\rangle \pm |11\rangle)$ or $|\Psi^\pm\rangle = \frac{1}{\sqrt{2}}(|01\rangle \pm |10\rangle)$, respectively.

* ("Bell", x, y) to return one of the Bell states due to the notation $\left|\beta_{xy}\right\rangle = [|0y\rangle + (-1)^x |1\bar{y}\rangle]/\sqrt{2}$, i.e.

$$|\beta_{00}\rangle = \left|\Phi^+\right\rangle = (|00\rangle + |11\rangle)/\sqrt{2}, \qquad |\beta_{01}\rangle = \left|\Psi^+\right\rangle = (|01\rangle + |10\rangle)/\sqrt{2},$$
$$|\beta_{10}\rangle = \left|\Phi^-\right\rangle = (|00\rangle - |11\rangle)/\sqrt{2}, \qquad |\beta_{11}\rangle = \left|\Psi^-\right\rangle = (|01\rangle - |10\rangle)/\sqrt{2}.$$

* ("Bell diagonal", $[p_I, p_x, p_y, p_z]$) to return a Bell diagonal state, i.e. the mixture of the two-qubit Bell states

$$\rho = p_I \left|\Phi^+\right\rangle\left\langle\Phi^+\right| + p_x \left|\Psi^+\right\rangle\left\langle\Psi^+\right| + p_y \left|\Psi^-\right\rangle\left\langle\Psi^-\right| + p_z \left|\Phi^-\right\rangle\left\langle\Phi^-\right|$$
$$= \frac{1}{2}\begin{pmatrix} p_I + p_z & 0 & 0 & p_I - p_z \\ 0 & p_x + p_y & p_x - p_y & 0 \\ 0 & p_x - p_y & p_x + p_y & 0 \\ p_I - p_z & 0 & 0 & p_I + p_z \end{pmatrix},$$

and where the (classical) weights p_i have to fulfill $\sum_i p_i = 1$. A Bell-diagonal state is separable if none of the weights is equal or larger 1/2.

* ("biseparable") to return the mixed 3-qubit state

$$\tilde{\rho}_{ABC} = \frac{1}{4}\left(I_8 - \sum_{i=1}^{4} |\psi_i\rangle\langle\psi_i|\right),$$

and where the $|\psi_i\rangle$ with $i = 1, \ldots, 4$ are given by $|01+\rangle$, $|1+0\rangle$, $|+01\rangle$ and $|---\rangle$, respectively. The biseparable states is known to exhibit genuine tripartite (bound) entanglement.

* ("Dicke", n, e) to return an n-qubit symmetric Dicke state with e excitations [37]

$$|n, e\rangle = \binom{N}{e}^{1/2} \sum_i P_i \left(|1_1, 1_2, \ldots, 1_e, 0_{e+1}, \ldots, 0_N\rangle\right),$$

and where the $\{P_i\}$ refer to the set of all distinct permutations of the qubits. The state $|n, 1\rangle$ is the same as the n-qubit W state.

* ("GHZ", n) to return the n-qubit Greenberger–Horne–Zeilinger (GHZ) state $|GHZ\rangle_n = \frac{1}{\sqrt{2}}(|00\ldots0\rangle + |11..1\rangle)$ [38].

* ("ground state", H) to return a qregister whose state represents the ground state of the given Hamiltonian matrix H. The ground state is obtained by direct diagonalization; this state is not unique if the ground state is degenerate.

* ("Ishizaka", $[p_1, p_2, p_3, p_4]$) or ("IH", $[p_1, p_2, p_3, p_4]$) to return the two-qubit Ishizaka–Hiroshima (IH) state

$$\rho_{\rm IH} = p_1 \left|\Psi^-\right\rangle\left\langle\Psi^-\right| + p_2 |00\rangle\langle 00| + p_3 \left|\Psi^+\right\rangle\left\langle\Psi^+\right| + p_4 |11\rangle\langle 11|$$
$$= \begin{pmatrix} p_2 & 0 & 0 & 0 \\ 0 & \frac{p_3 + p_1}{2} & \frac{p_3 - p_1}{2} & 0 \\ 0 & \frac{p_3 - p_1}{2} & \frac{p_3 + p_1}{2} & 0 \\ 0 & 0 & 0 & p_4 \end{pmatrix}$$

for the (real) eigenvalues $\{p_i; \sum_i p_i = 1$, and with $p1 \geq p_2 \geq p_3 \geq p4\}$. For the IH state, the entanglement of formation and the negativity cannot be increased by local unitaries on the two qubits. Moreover, all Werner states are known to belong to the IH class. The concurrence of the IH-states with rank ≤ 3 is given by $\mathcal{C}_{\rm IH} = p_1 - p_3 - 2\sqrt{p_2\, p_4}$; for rank 4, this relation is known to be fulfilled numerically [39].

* ("isotropic", F) to return an $U \otimes U^*$-invariant (two-qubit) mixed isotropic state

$$\rho_{iso} = \frac{1 - F}{3}(I - \left|\Psi^+\right\rangle\left\langle\Psi^+\right|) + F \left|\Psi^+\right\rangle\left\langle\Psi^+\right|$$

that is composed as a mixture of the two-qubit Bell state $\left|\Psi^+\right\rangle$ and a (unnormalized) two-qubit maximally mixed state $I = I_4$, and where $F = \left\langle\Psi^+\right| \rho_{iso} \left|\Psi^+\right\rangle$ denotes the fidelity of ρ_{iso} and $\left|\Psi^+\right\rangle$. For $F \leq \frac{1}{2}$, these density matrices are known to be separable

* ("Smolin") to return the four-qubit bound entangled state [40]

$$\rho = \frac{1}{4}\sum_{x,y=0,1} \left|\Psi_{xy}\right\rangle\left\langle\Psi_{xy}\right| \otimes \left|\Psi_{xy}\right\rangle\left\langle\Psi_{xy}\right|$$
$$= \frac{1}{4}\left(\left|\Phi^+\right\rangle\left\langle\Phi^+\right| \otimes \left|\Phi^+\right\rangle\left\langle\Phi^+\right| + \left|\Phi^+\right\rangle\left\langle\Phi^+\right| \otimes \left|\Phi^+\right\rangle\left\langle\Phi^+\right|\right).$$

Although the Smolin state is bound entangled, it violates maximally the Clauser–Horne–Shimony–Holt-type Bell inequality for four qubits.

* ("thermal", H, T) to return the thermal state $\rho_{\rm th} = e^{-H/T}/{\rm Tr}(e^{-H/T})$, where H is the Hamiltonian matrix and T the temperature, and where the Boltzmann constant is assumed to be $k = 1$.

* ("W", n) to return the n-qubit W state

$$|W\rangle_n = \frac{1}{\sqrt{n}}\left(|00\ldots01\rangle + |00\ldots10\rangle + \cdots + |10\ldots00\rangle\right).$$

The W states are maximally robust under particle loss, and they are immune against global dephasing.

* ("Werner", p) to return an $U \otimes U$-invariant (two-qubit) mixed Werner state [41]

$$\rho_{\mathrm{W}} = p \left|\Psi^-\right\rangle\left\langle\Psi^-\right| + (1-p)\,\frac{1}{4} I_4$$

with probability $0 \le p \le 1$. Werner states are separable for $p \le 1/3$ and entangled otherwise. The definition of ρ_{W} is often generalized by including maximally entangled state (MES) instead of the singlet Bell state $\left|\Psi^-\right\rangle$.

(e) Maximally mixed and entangled states

* ("maximally entangled", k_{even}) to return an k-qubit qregister() in the maximally entangled state

$$|\Phi\rangle_{AA'} = \frac{1}{\sqrt{d_A}} \sum_{i=1}^{d_A} |i\rangle_A \otimes |i\rangle_{A'}$$

that is symmetric with regard to its subspaces $\mathbb{C}^{d_A} \otimes \mathbb{C}^{d_{A'}}$, and where $d_A = d_{A'} = 2^{k/2}$.

* ("maximally mixed", n) to return an n-qubit qregister() in the totally mixed state $\rho = \frac{1}{d} I_d$, where $d = 2^n$ and I_d is the identity matrix of dimension d.

(f) Graph states

* ("graph", n, [[i, j], [k, l], ...]) to generate a pure n-qubit graph state with edges E as defined by the (pairs of) qubit indices [[i,j], [k,l], ...]. The n-qubit graph state is then given as

$$|G\rangle = \prod_{(a,b)\in E} U_{\mathrm{CZ}}^{\{a,b\}} |+\rangle^{\otimes n},$$

by applying a controlled-Z gate to every edge.

* ("graph: chain", n) to generate a n-qubit chain graph state where only neighbors $(i,\ i+1)$ are connected to each other.
* ("graph: ring", n) to generate a n-qubit ring graph state where only neighbors $(i,\ i+1)$ are connected but with the additional $(n,\ 1)$ edge to form a ring.

References

[1] C.H. Bennett, G. Brassard, C. Crepeau, R. Jozsa, A. Peres, W.K. Wootters, Phys. Rev. Lett. 70 (1993) 1895.
[2] D. Boschi, S. Branca, F. De Martini, L. Hardy, S. Popescu, Phys. Rev. Lett. 80 (1998) 1121.
[3] L.-M. Duan, M.D. Lukin, J.I. Cirac, P. Zoller, Nature 414 (2001) 413.
[4] C. Wang, F.-G. Deng, Y.-S. Li, X.-S. Liu, G.L. Long, Phys. Rev. A71 (2005) 044305.
[5] Y.L. Lim, A. Beige, L.C. Kwek, Phys. Rev. Lett. 95 (2005) 030505.
[6] M.A. Nielsen, I.L. Chuang, Quantum Computation and Quantum Information, Cambridge University Press, Cambridge, 2000.
[7] G. Toth, Comput. Phys. Comm. 179 (2008) 430.
[8] F. Tabakin, B. Julia-Diaz, Comput. Phys. Commun. 182 (2011) 1693.
[9] 2013. http://www.quantiki.org/wiki/List_of_QC_simulators.
[10] T. Radtke, S. Fritzsche, Comput. Phys. Commun. 173 (2005) 91.
[11] T. Radtke, S. Fritzsche, Comput. Phys. Commun. 176 (2007) 617;
T. Radtke, S. Fritzsche, Comput. Phys. Commun. 179 (2008) 647.
[12] J.F. Clauser, M.A. Horne, A. Shimony, R.A. Holt, Phys. Rev. Lett. 23 (1969) 880.
[13] D. Deutsch, R. Jozsa, Proc. R. Soc. A439 (1992) 553.
[14] P.W. Shor, SIAM J. Sci. Stat. Comput. 26 (1997) 1484.
[15] L.K. Grover, Phys. Rev. Lett. 79 (1997) 325.
[16] B. Butscher, H. Weimer, http://www.libquantum.de/.
[17] F.L. Marquezino, R. Portugal, Comp. Phys. Commun. 179 (2008) 359.
[18] T. Radtke, S. Fritzsche, Comput. Phys. Commun. 175 (2006) 145.
[19] T. Radtke, S. Fritzsche, Comput. Phys. Commun. 181 (2010) 440.
[20] http://tph.tuwien.ac.at/~oemer/qcl.html.
[21] S. Fritzsche, Comput. Phys. Commun. 103 (1997) 51;
S. Fritzsche, Comput. Phys. Commun. 111 (1998) 167;
S. Fritzsche, Comput. Phys. Commun. 139 (2001) 314.
[22] S. Fritzsche, Int. J. Quant. Chem. 106 (2006) 98.
[23] A. Gilchrist, N.A. Langford, M.A. Nielsen, Phys. Rev. A71 (2005) 062310.
[24] S. Luo, Q. Zhang, Phys. Rev. A69 (2004) 032106.
[25] B. Schumacher, Phys. Rev. A54 (1998) 2614.
[26] V. Vedral, Rev. Modern Phys. 74 (2002) 197.
[27] W.K. Wootters, Phys. Rev. Lett. 80 (1998) 2245.
[28] C.H. Bennett, D.P. DiVincenzo, J.A. Smolin, W.K. Wootters, Phys. Rev. A54 (1996) 3824.
[29] G. Vidal, R.F. Werner, Phys. Rev. A65 (2002) 032314.
[30] A.R.R. Carvalho, F. Mintert, A. Buchleitner, Phys. Rev. Lett. 93 (2004) 230501.
[31] A. Wong, N. Christensen, Phys. Rev. A63 (2001) 044301.
[32] A. Jamiołkowski, Rep. Math. Phys. 3 (1972) 275.
[33] M. Ziman, V. BuZek, Phys. Rev. A72 (2005) 052325.
[34] T. Jennewein, G. Weihs, J.-W. Pan, A. Zeilinger, Phys. Rev. Lett. 88 (2001) 017903.
[35] X.-S. Ma, S. Zotter, J. Kofler, R. Ursin, T. Jennewein, C. Brukner, A. Zeilinger, Nat. Phys. 8 (2012) 412.
[36] A. Peres, J. Modern Opt. 47 (2000) 139.
[37] L. Mandel, E. Wolf, Optical Coherence and Quantum Optics, Cambridge Univ. Press, Cambridge, 1995.
[38] D.M. Greenberger, M.A. Horne, A. Shimony, A. Zeilinger, Amer. J. Phys. 58 (1990) 1131.
[39] S. Ishizaka, T. Hiroshima, Phys. Rev. A62 (2000) 022310.
[40] J.A. Smolin, Phys. Rev. A63 (2001) 032306.
[41] R.F. Werner, Phys. Rev. A40 (1989) 4277.