David Butler
University of California, Los Angles
dwbutler@ucla.edu

Valid Input & Error Detection

.Any computer program expects certain kinds of input, and given the right type of input, needs to check that this input is valid for the purposes of the program. For example, a program that parses a file might expect a file name as input, and given a valid string, needs to check that the string refers to an existing file that can be opened. The purpose of our program is, given a set of generators, to output them in graphical form. To understand what constitutes valid input to our program, it is necessary to understand a little bit of group theory.

Group Theory and its Relation to Quantum States

.An n-qubit quantum state may be compactly represented using its generators. In group theory, the *generators* of a group is the subset of a group, the members of which can be used to generate all the other members of the group. In other words, by successive applications of the generating elements in a group, you can generate all the elements present in the larger group.

.Group theory is used in quantum mechanics to compactly represent quantum states. This is done by representing the quantum state as a group of *unitary operators*. (If quantum states are thought of as unit length vectors in complex Hilbert space, unitary operators may be thought of as matrices which rotate quantum states.) The group may be of arbitrary size, but is most compactly represented using the generating group. The members of such a group are termed generators. Most commonly, members of the *Pauli group* are used as the generators to describe a quantum state:

$$\{.[i]I, .[i]X, .[i]Y, .[i]Z\}$$

where the brackets [] indicate optional components. (Please see the references for material that explains in more detail the definition and meaning of the operators X, Y, and Z.) In principle, therefore, there are 16 possible generators in the Pauli group. Any composition of these could serve as valid basic input to the program. In practice, however, there are extra conditions imposed by the limited purpose of our program. For example, the multiplicative factors (.[*i*]) are ignored for the purposes of graphing the generators, for reasons that will become clear later. Additionally, the operator -I is not allowed (for another reason which will be explained later). Y can be generated using X and Z, so we don't use it. Therefore, for our program, the only valid input is a composition of the following operators:

$$\{I, X, Z\}$$

Error Detection on Input

.Given an arbitrary group of generators, how do we know that such a group will actually work? It is not enough that we be given generators composed of members of {I, X, Z}. We also require that the generators actually generate a valid, nontrivial quantum state. That is to say, we need to check

whether well-formed generators actually generate something. It turns out that there are only three simple conditions that need to be imposed on a generating group S to guarantee this:

1..-I must not be an element.
2..The elements of S must commute.
3..The elements of S must be linearly independent.

Condition number one is guaranteed by our restriction on the input to {I, X, Z}. Conditions two and three are easily checked by using what we call a *check matrix*. Given n generators of size n, the check matrix is of size n x 2n (n rows and 2n columns). The matrix can be thought of as two halves. The left half contains information about where the X operators are located in the generators. The right half contains information about where the Z operators are located in the generators. A 1 in the left half of the matrix indicates an X operator in that position, a 1 in the right half indicates a Z in that position. A zero in both halves of the matrix indicate an I operator in that position. For example:

The second condition specifies that the elements of S must commute. This means that, for example, that $S_1S_2|k\rangle=S_2S_1|k\rangle$. This can be checked using the check matrix. Suppose that the check matrix of S is g. Then, the elements of S commute if and only if:

$$gLg^T=0$$

where T indicates the transpose of a matrix, and L is defined as follows:

$$L=\begin{bmatrix} 0 & I \\ I & 0 \end{bmatrix}$$

The third condition holds if and only if the rows of the check matrix are linearly independent.

Implementation

.While the GUI of our program was written in Perl, the mathematical portion was written in C. We used a library called Gandalf in order to utilize its built-in linear algebra functions. Input to our program is in the form of a matrix. Each row is one generator, and each column is a position in the generator. Only characters in the set {i,I,x,X,z,Z} are allowed. Once inputted, the generators are converted into a check matrix. This check matrix is passed to a function that checks its validity. This is the function that I wrote. Valid input characters are guaranteed by the Perl GUI, which only allows valid input characters. From there, a well-formed check matrix is guaranteed by a function in the C code. This well-formed check matrix is passed to my function, and the generators checked for validity.

.Conditions two and three are the only conditions that actually need to be checked. Condition three

is not vital, and Gandalf had no functions to check the linear independence of the rows of a matrix. I therefore chose not to perform this check. Condition two is therefore the only condition my function checks.

.The first step is to generate the nxn L matrix. The only way to fill a matrix with values in Gandalf is to either put the same value in all slots, or enumerate what specific value ought to go in each slot. I had to use the second method to create a general-size L matrix. Given n, the height of the check matrix, my function uses two nested for loops to iterate over all slots in the L matrix. Every slot has a value of zero unless we are on the diagonal of the lower left or upper right of the matrix.

.Once the L matrix is generated, the program uses Gandalf to perform matrix multiplication and transposition. The result of these operations is an nxn matrix. If this matrix is filled with zeros, then the test passes. Otherwise it fails. The program must iterate through all slots in the matrix to check that they are all zero.