py$_q codeDocumentation$

Release 0.0

Ben Criger

May 04, 2014

CONTENTS

Contents:

CHAPTER ONE

SIMULATION

1.1 Class Reference

class `py_qcode.`**`Simulation`**(*lattice*, *error_model*, *code*, *decoder*, *n_trials*)

Simulation is the top-level class for py_qcode, the user is meant to set up, execute and save results using these objects.

Parameters

- **error_model** (`py_qcode:ErrorModel`, input) – A description of the errors to be applied independently to the qubits of *lattice*.
- **code** (`py_qcode.ErrorCorrectingCode`, input) – An error-correcting code which translates errors on a lattice to syndromes on the dual lattice.
- **decoder** (`py_qcode.Decoder`, input) – A protocol for inferring errors given syndromes.
- **n_trials** (*integer, input*) – a number of simulations to be performed in series. This can be used to organize batch jobs so that one can submit more than one simulation per job.
- **true_coset** (*str, output*) – The actual coset to which the random error applied in the simulation belongs.
- **inferred_coset** – The coset assigned by the error-correcting code during the simulation.

`run`()

The main routine in this library, follows the recipe *n_trials* times in series:

•Apply the error model to the primary lattice, assigning values to the *error* attributes of the `py_qcode.Point` objects within.

•Obtain the true coset of the error with respect to the `py_qcode.ErrorCorrectingCode` being used.

•Perform a measurement, using the attributes of the error-correcting code to generate syndromes on the dual lattice.

•Infer the coset of the error by acting the decoder on the dual lattice.

•Record both cosets.

`save`()

CHAPTER

TWO

LATTICES

2.1 Class Reference

class `py_qcode.`**`Point`**(*coords*, *error=None*, *syndrome=None*)

Represents a point in two or three dimensions. Normally, I'd use a `namedtuple`, because I'm not a psychopath, but I want to use default arguments without getting too fancy. Each point can also contain a value denoting the error which an `ErrorModel` has applied to that point, and/or a syndrome which results from measurement of an `ErrorCorrectingCode` on the lattice.

Parameters

- **coords** (*tuple of ints, length 2 or 3*) – co-ordinates of the point in question.
- **error** (*any*) – A value which denotes an error. An `ErrorCorrectingCode` must check that this value corresponds to an operator which can be translated into a syndrome.
- **syndrome** (*any*) – A value which denotes an syndrome. A `Decoder` must check that this value corresponds to an operator which can be translated into a syndrome.

class `py_qcode.`**`Lattice`**(*sz_tpl*, *is_3D=False*, *closed_boundary=True*)

A collection of points. Superclass to `SquareLattice`, `SquareOctagonLattice`, `UnionJackLattice`, whatever other convenient lattices I put in.

Represents a 2D/3D lattice of points with integer co-ordinates on which a stabilizer code can be defined.

Parameters

- **sz_tpl** (*tuple, length 2 or 3, containing integers.*) – linear dimensions of the lattice.
- **is_3D** (*bool*) – indicates whether the lattice is to be two- or three- dimensional.
- **closed_boundary** (*bool*) – Indicates whether to identify the Nth co-ordinate with the zeroth co-ordinate in every dimension.

class `py_qcode.`**`SquareLattice`**(*sz_tpl*, *is_3D=False*, *closed_boundary=True*, *rough_sides=('f', 'b')*)

Represents a lattice in which qubits are placed on the edges of a grid of squares.

Parameters **rough_sides** (*tuple of strings*) – Denotes which, if any, of the sides of the lattice are to have 'rough' boundary conditions. Values in `rough_sides` must be drawn from `['u', 'd', 'r', 'l', 'f', 'b']` (up, down, left, right, front, back).

class `py_qcode.`**`SquareOctagonLattice`**(*sz_tpl*, *is_3D=False*, *closed_boundary=True*)

Represents a lattice in which qubits are placed on the corners of squares and octagons.

class `py_qcode.`**`UnionJackLattice`**(*sz_tpl*)

Represents a lattice in which qubits are placed on the intersections of the diagonals of squares, as well as their corners.

CHAPTER

THREE

ERROR CORRECTING CODES

3.1 Class Reference

class `py_qcode.`**`ErrorCorrectingCode`**`(`*arg*`)`

An error-correcting code, for the purpose of this module, is a rule for taking errors on sets of points, and turning them into discrete-valued syndromes which are interpreted by the decoder. Normally, we would immediately make the restriction to stabilizer codes which return binary-valued syndromes, but we want to make room for codes which correct non-Pauli errors, and return fuzzy syndromes.

Parameters

- **primal_lattice** (`py_qcode.Lattice`) – The lattice on which the qubits live.
- **dual_lattice** (`py_qcode.Lattice`) – The lattice on which the parity checks live.
- **parity_check** (*function*) – A rule for mapping errors on the primal lattice to measurement results on the dual lattice.

class `py_qcode.`**`StabilizerCode`**`(`*arg*`)`

subclass of ErrorCorrectingCode for which syndromes are determined by commutation /anti-commutation with a Pauli stabilizer.

CHAPTER

FOUR

ERROR MODELS

4.1 Class Reference

class `py_qcode.`**`ErrorModel`**(*prob_op_list*)

Wraps a list of tuples corresponding to a discrete probability and an operator. This assumes independent identically-distributed noise, though not necessarily Pauli.

Parameters **prob_op_list** (*list*) – A list of probabilites and associated operators. The probabilites are floats which must sum to 1 to within 10^{-12}. The operators are represented by strings which must be drawn from the list of acceptable operators *['I','X','Y','Z','H','P']* Each pair (a probability and its associated operator) is stored in a tuple.

class `py_qcode.`**`DepolarizingModel`**(*p*)

The depolarizing model applies the identity with probability $1 - p$, and each of the single qubit Pauli operators X, Y, and Z with probability $\frac{p}{3}$.

Parameters **p** (*float*) – A probability, between 0 and 1.

CHAPTER

FIVE

DECODERS

5.1 Class Reference

class `py_qcode.`**`Decoder`**`(`*arg*`)`
    docstring for Decoder

class `py_qcode.`**`MWPMDecoder`**`(`*arg*`)`
    docstring for MWPMDecoder

class `py_qcode.`**`RGBPDecoder`**`(`*arg*`)`
    docstring for RGBPDecoder

class `py_qcode.`**`BHRGDecoder`**`(`*arg*`)`
    docstring for BHRGDecoder

CHAPTER

SIX

INTRODUCTION

This library is concerned with simulating the decoding of topological quantum error-correcting codes. This is useful for determining their threshold error rates and logical error rates in the regime where useful error correction can be performed. In order to simulate the act of decoding, the following components are necessary:

- **Input/Output**: Routines to input necessary values such as the name of the lattice geometry, size, error-correcting code to be used, etc., and save the resulting output to a file.
- **Lattice**: A set of points on which the qubits and stabilizers of the code are defined.
- **Error Model**: A set of Gottesman-Knill-compliant operators and probabilities with which they are applied to the qubits in the lattice.
- **Error-Correcting Code**: A set of stabilizer generators, along with their support, which produces error syndromes given an error.
- **Decoder**: A rule for inferring the original error, given a syndrome.

CHAPTER

SEVEN

INDICES AND TABLES

- *genindex*
- *modindex*
- *search*