Python Implementation of Stabilizer Algorithms

Patrick Rall, Iskren Vankov - March 14, 2016

Motivation

- Goal: Implement stabilizer algorithms from the appendix in python
- Why python and not C++ as originally planned?
  - We have better familiarity with python and its linear algebra library numpy
  - Our first implementation should focus on *correctness* rather than efficiency
  - We can make graphs and plots and design sophisticated tests to ensure that no bugs are present
- Once we have a correct implementation we can copy the code to C++, focusing on optimization rather than algorithmic details

Progress so far

- Implementation of $q(x)$ (eq. 42) and state vector coefficient extraction (eq. 46)
- Helper functions: update D, J using (eqs. 48, 49), update Q, D using (eqs. 51, 52)
- The Shrink routine and Shrink* routine
- The RandomStabilizerState function

Questions

- Do the distributions below look correct?
- What other tests can you think of to ensure that the code is correct?
  We already check $G\bar{G}^T = I \pmod 2$.
- Mini project idea: Distribution of MPS Schmidt rank χ for stabilizer states

Next steps

- Implement the remaining routines in the appendix
- Begin implementing the main quantum circuit simulator
- Begin outlining C++ code

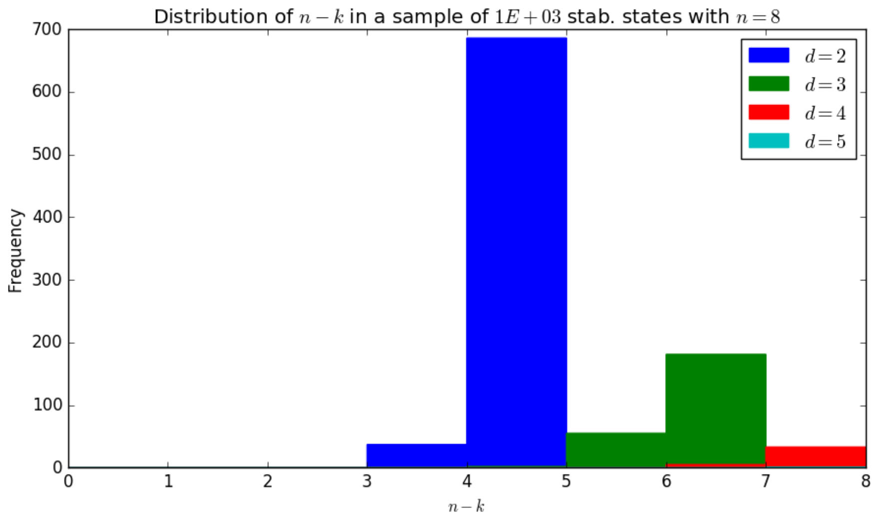

Figure 1: Distribution of sampled $n-k$. In RandomStabilizerState, d is sampled from a probability distribution and k is initially set to $n-d$. Then the Shrink* routine is applied several times, potentially reducing k and increasing $n-k$ a little. Separate colors show the original d values so we can see how many times Shrink* was applied.

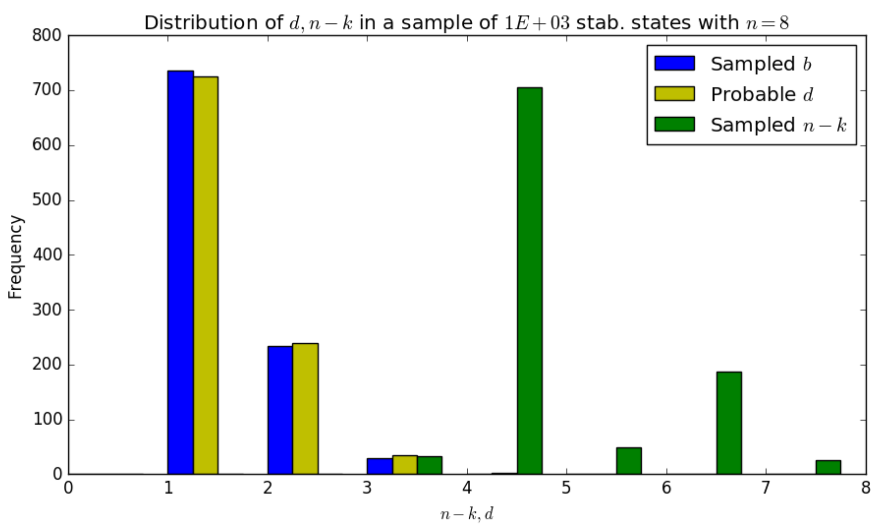

Figure 2: Distributions of sampled b and $n-k$, and the scaled probability distribution $P(d)$. We see sampled d and $P(d)$ are very close, but $n-k$ deviates because of the applications of Shrink*. If Shrink* was never applied then the distributions should all be equal.