Spin Coherent States Notes

Rafael Josh

May 2020

1 Introduction

Spin coherent states are coherent states of a spin system rather than a quantum harmonic oscillator. The coherent state $|a\rangle$ is an eigenstate of the lowering operator $\hat{a}$ with an eigenvalue a such that $\hat{a}|a\rangle = a|a\rangle$. While this is still true for spin coherence, we instead are looking at spin states which are more analogous to classical spin. The main purpose of spin coherent states is to construct the spin coherent state path integral. We will do so for our purposes of spin-$\frac{1}{2}$ particles

2 Defining Spin Coherent States

We can consider a spin with a fixed angular momentum quantum number s $\in$ {0, 1/2, 1, 3/2,...}. We know from the properties of spin operators that $\langle ss|\hat{S_x}|ss\rangle$=0, $\langle ss|\hat{S_y}|ss\rangle$=0, and $\langle ss|\hat{S_z}|ss\rangle$=s. Thus, the expectation value of the total spin operator $\hat{\mathbf{S}}$ in the state $|s\rangle$ is s$\mathbf{e}_z$. We can define a vector $\mathbf{s}$ to have magnitude s and direction (θ, ϕ). The spin coherent state $|\mathbf{s}\rangle$ is the state obtained by rotating $|ss\rangle$ by θ and ϕ about the y and z axes.

$$|\mathbf{s}\rangle = |s\theta\phi\rangle = e^{i\theta\hat{S_y}} e^{i\phi\hat{S_z}}\,|ss\rangle \tag{1}$$

Using a Wigner-D Matrix (Kim) we can relate $|\mathbf{s}\rangle$ to $|sm\rangle$:

$$|s\theta\phi\rangle = \sum_{m=-s}^{m=s} \sqrt{\frac{(2s)!}{(s+m)!(s-m)!}}\ (cos\frac{\theta}{2})^{s+m}\ \ (sin\frac{\theta}{2})^{s-m}\ \ e^{-im\phi}\ \ |sm\rangle \tag{2}$$

Since we have restricted ourselves to spin-$\frac{1}{2}$ particles, we can simplify Equation (2) to a sum of two elements for $\left|\frac{1}{2}, \frac{1}{2}\right\rangle$ and $\left|\frac{1}{2}, \frac{-1}{2}\right\rangle$.

$$|s\theta\phi\rangle = \begin{pmatrix} cos\frac{\theta}{2} \\ sin\frac{\theta}{2}\ e^{i\frac{\phi}{2}} \end{pmatrix} \equiv |\theta\phi\rangle \tag{3}$$

By extending Equation (2) we can show the inner product of two different values of θ and ϕ becomes

$$\langle\theta\phi|\theta'\phi'\rangle = cos\frac{\theta}{2}cos\frac{\theta'}{2} + sin\frac{\theta}{2}sin\frac{\theta'}{2}e^{\frac{i}{2}(\phi-\phi')} \tag{4}$$

This represents the overlap in the two spin coherent states and is maximized to 1 when $|\theta\phi\rangle = |\theta'\phi'\rangle$. We can form the identity matrix by integrating over the entire solid angle of the sphere:

$$\hat{1} = \frac{1}{2\pi}\int d\Omega\, |\theta\phi\rangle\, \langle\theta\phi| = \frac{1}{2\pi}\int d\theta\ d\phi\ sin\theta\, |\theta\phi\rangle\, \langle\theta\phi| \tag{5}$$

We can apply the identity to every time step in the wavefunction. By this, we are referring to the structure that our wavefunction takes as a series of transformations for each time step we would like to measure. Here are the identities applied to each of these steps in the wavefunction:

$$|\psi(t)\rangle = \ \hat{1}\ e^{-iH\delta}\ ...\ \hat{1}\ e^{-iH\delta}\ \hat{1}\ \ |\psi(0)\rangle =$$

$$= \tfrac{1}{(2\pi)^N}\int d\Omega_N\ d\Omega_{N-1}\ ...\ d\Omega_1\, |\theta_N\phi_N\rangle\, \langle\theta_N\phi_N|\, e^{-iH\delta}\, |\theta_{N-1}\phi_{N-1}\rangle\ ...\ \ |\theta_1\phi_1\rangle\, \langle\theta_1\phi_1|\, e^{-iH\delta}\, |\psi(0)\rangle \tag{6}$$

Here, $e^{-iH\delta}$ is the unitary time evolution operator, and we have chosen the Hamiltonian H to be $\frac{-b}{2\hbar}\sigma_y$ where σ_y is the Pauli-y matrix and we have chosen b=1. In the time evolution operator, δ represents the time-dependence but it is divided by the number of time steps N that we use as input to our matrix elements ($\delta = \frac{t}{N}$). Our matrix elements $\langle\theta_j\phi_j|\, e^{-iH\delta}\, |\theta_{j-1}\phi_{j-1}\rangle$ can be calculated as matrix exponentials and represents the time evolution or one "path" from one state to another. Each state $|\theta_{j-1}\phi_{j-1}\rangle$ is two matrix exponentials as in Equation (1) and is applied to the state at $\theta = 0$, $\phi = 0$ as made by Equation (3). Putting it all together, the matrix element looks like: $\langle 0|\, e^{\frac{-i}{2}\theta_j\sigma_y}e^{\frac{-i}{2}\phi_j\sigma_z}e^{\frac{-i}{2}H\delta}e^{\frac{i}{2}\theta_{j-1}\sigma_z}e^{\frac{i}{2}\phi_{j-1}\sigma_z}\, |0\rangle$. That is the expectation value of all the above in the $|0\rangle$ state. In terms of code, this just means we must surround our matrix exponentials with the vector (1,0). Here, we have also set $\hbar = 1$.

We have implemented equation (6) in Mathematica by making two functions. One for final state which represents the overlap between $\langle 0|\, or\, \langle 1|$ and $|\theta_N\phi_N\rangle$.

This is simply two matrix exponentials surrounded by the (1,0) vector for our initial $|0\rangle$ state and either (1,0) or (0,1) for whichever final state we choose. The other function is for the product of $\langle\theta_N\phi_N| e^{-iH\delta} |\theta_{N-1}\phi_{N-1}\rangle$ ... $\langle\theta_2\phi_2| e^{-iH\delta} |\theta_1\phi_1\rangle$. This function depends on t due to the $e^{-iH\delta}$ term, and takes in a list of two thetas and two phis for the j and j-1 sides of the matrix element. We can multiply `overlap*Product[matelem]*matelem(0,0,`θ_1`,`ϕ_1`)` (where the last matelem is the element $\langle\theta_1\phi_1| e^{-iH\delta} |\psi(0)\rangle$) to create our baseline function we call `f`. This baseline function takes in a value for t from the matrix elements function, the number of timesteps N, a final state, a list of thetas the length of N, and a list of phis the length of N. Our value for time is typically chosen to be $\frac{\pi}{2}$ since we want to be able to compare many different methods and we are not integrating over it. To validate that our baseline function is working we integrate over all theta and phi possibilities in a sphere and check with classically expected coefficients at t=$\frac{\pi}{2}$. These classically expected coefficients we define as

$$c_m = \int_\Omega D(\Omega) f_m(\Omega) = \tag{7}$$

$$c_m = \int_{\theta=0}^{\pi}\int_{\phi=0}^{2\pi} d\theta_1 d\phi_1 ... d\theta_n d\phi_n * f(t,n,m,[\theta_1,\theta_2,...,\theta_n],[\phi_1,\phi_2,...,\phi_n]) * sin(\theta_1) sin(\theta_2) ... sin(\theta_n) \tag{8}$$

Where,

$$f_m = \frac{1}{(2\pi)^N} \langle m|\theta_N\phi_N\rangle \langle\theta_N\phi_N| e^{-iH\delta} |\theta_{N-1}\phi_{N-1}\rangle \ ... \ |\theta_1\phi_1\rangle \langle\theta_1\phi_1| e^{-iH\delta} |\psi(0)\rangle \tag{9}$$

Note that this is the same as Equation (6) with definite boundaries. Here, m is the final state and decides c_0 or c_1 which decides whether we return a value of a cosine or a sine respectively. For t=$\frac{\pi}{2}$ we expect $c_0 = \frac{\sqrt{2}}{2}$ and $c_1 = -\frac{\sqrt{2}}{2}$.

3 Monte Carlo Sampling

For the purpose of modeling spin coherent states, it is useful to employ Monte Carlo random sampling to add these possibilities in a different way. The simplest way to incorporate Monte Carlo is creating a list of random theta and phi values in our baseline function so that we can create a new set of random values each time the function is called. So rather than taking a list of thetas and phis as a parameter to the function, we can simply make the lists inside the function. The list of thetas is a list of the arc cosine of random numbers between -1 and 1 due to our necessity of even sampling between these end points. The list of phis is a list of random numbers between 0 and 2π. These lists of random numbers make up our paths that we are sampling the result of when plugged into our `f`. When we sample many of these paths, we expect the mean of our dataset to approach $\frac{\sqrt{2}}{2}$, simulating the integral that we evaluated theoretically and numerically in a Monte Carlo style.

To verify our Monte Carlo method we make a function named `mcflist` to sample over `f` the amount of times of which is entered as a parameter to `mcflist`. We then take the mean of `mcflist` with the `Mean` function in Mathematica to get the classically expected coefficient $\frac{\sqrt{2}}{2}$ at t=$\frac{\pi}{2}$ as before.

We also analyze standard deviations of `mcflist` at increasing number of timesteps to get an exponentially growing function. We do this using Mathematica's `StandardDeviation` function on `mcflist`. We then linearize the function by taking the Log of the exponential and find the slope of the linearized function so we can later compare it to our Metropolis-Hastings method of adding spin paths.

While we are analyzing the standard deviation, it is important to note that we are actually analyzing something proportional to the standard deviation which we call

$$E = s\,\sqrt{n_{samples}} \tag{10}$$

Where

$$s = \frac{\sigma}{\sqrt{n_{samples} - 1}} \tag{11}$$

E is simply the standard deviation of the data, σ, divided by the square root of $n_{samples} - 1$ and multiplied by the square root of $n_{samples}$. So, in total, we will graph the $log(E)$ for our analysis.

Then to fit a line onto the data, we use Mathematica's `LinearModelFit` function which prints out a simple line equation that we can then graph. From the `LinearModelFit` function we can also extract a standard error on the fit with "Parameter Table". This is a measure of how close the projected line with exact slope comes to modeling the data. The error is calculated by Mathematica as:

$$error_{fitted} = \sqrt{\frac{\frac{1}{n-2}\sum_{i=1}^{n}\epsilon_i^2}{\sum_{i=1}^{n}(x_i - \bar{x})}} \tag{12}$$

Where ϵ is the residual of each point, $\bar{x}$ is the average of the number of timesteps, and x_i is the timestep value at that index. Our value for this was 0.004624 and shows how close each point was to the fitted line.

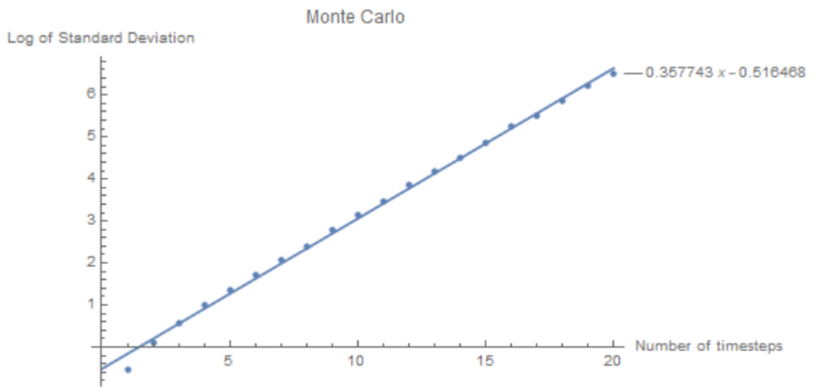


Figure 1: A graph showing the results of our function `f` sampled as Monte Carlo and taking the Log of Standard Deviation of the results versus the number of timesteps used

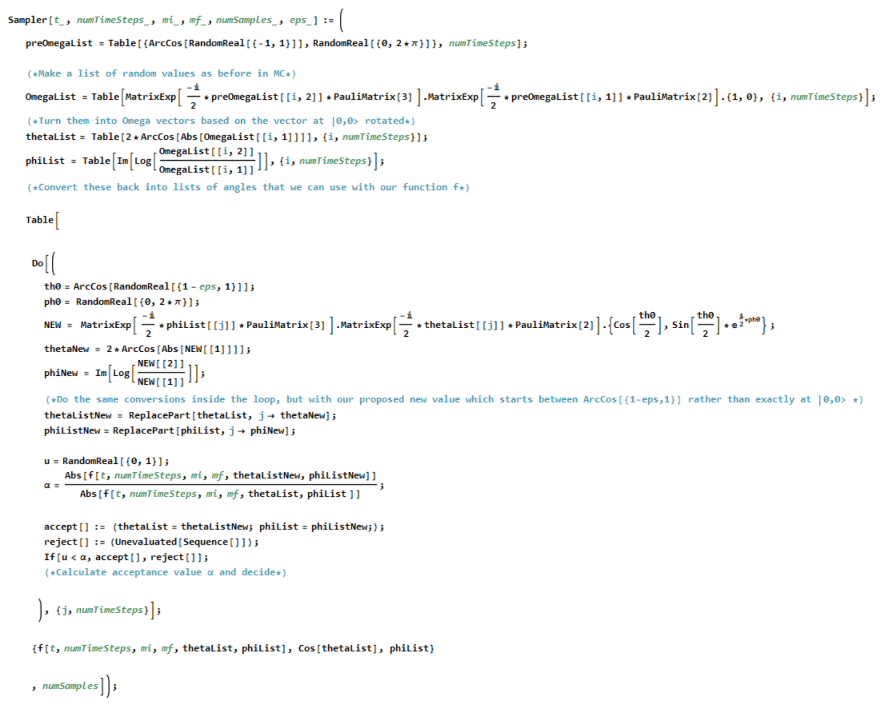
4 Metropolis-Hastings Sampling

Metropolis-Hastings algorithm is a different way of random sampling which involves both Monte Carlo and Markov Chains. The process is relatively simple. We start with some arbitrary starting point and have some function which suggests a next step which is typically decided by a gaussian distribution, but not in our case. Finally, at each step some acceptance ratio is used to accept or reject the suggestion and then continue the loop onto the next suggestion. The acceptance ratio we us is given by:

$$\alpha = Min(1, \frac{|f(\vec{\Omega}_{new})|}{|f(\vec{\Omega}_{old})|}) \tag{13}$$

Where $\vec{\Omega}_{old}$ is the list decided by the last iteration of the loop and $\vec{\Omega}_{new}$ is that same list but with the new suggestion in place of the index in the list under consideration.

We begin by rotating from $\theta = 0, \phi = 0$ by randomly chosen θ and ϕ on the sphere. We must do this using the matrix exponential as in Equation (1) to create an Omega for each point and re-extract a list of thetas and phis which we will use in our original `f`. For the function which makes a suggestion to the next step, we chose to sample somewhere near the north pole of the sphere meaning $cos(\theta) \approx 1$ and to rotate this by the θ and ϕ at the point in the list that we are evaluating. How close to the north pole is decided on by a value in the parameters called `eps`, short for epsilon. The larger `eps`, the greater range of possible starting position "near the north pole". This suggestion will direct us to somewhere else in the sphere and then we decide whether we accept this suggestion by acceptance ratio. This function is displayed and annotated below.



```
Sampler[t_, numTimeSteps_, mi_, mf_, numSamples_, eps_] := (
  preOmegaList = Table[{ArcCos[RandomReal[{-1, 1}]], RandomReal[{0, 2*π}]}, numTimeSteps];

  (*Make a list of random values as before in MC*)
  OmegaList = Table[MatrixExp[(-i)/2 * preOmegaList[[i, 2]] * PauliMatrix[3] ].MatrixExp[(-i)/2 * preOmegaList[[i, 1]] * PauliMatrix[2]].{1, 0}, {i, numTimeSteps}];
  (*Turn them into Omega vectors based on the vector at |0,0> rotated*)
  thetaList = Table[2*ArcCos[Abs[OmegaList[[i, 1]]]], {i, numTimeSteps}];
  phiList = Table[Im[Log[OmegaList[[i, 2]] / OmegaList[[i, 1]]]], {i, numTimeSteps}];
  (*Convert these back into lists of angles that we can use with our function f*)

  Table[

   Do[(
     th0 = ArcCos[RandomReal[{1 - eps, 1}]];
     ph0 = RandomReal[{0, 2*π}];
     NEW = MatrixExp[(-i)/2 * phiList[[j]] * PauliMatrix[3] ].MatrixExp[(-i)/2 * thetaList[[j]] * PauliMatrix[2]].{Cos[th0/2], Sin[th0/2] * e^(i/2*ph0)};
     thetaNew = 2*ArcCos[Abs[NEW[[1]]]];
     phiNew = Im[Log[NEW[[2]] / NEW[[1]]]];
     (*Do the same conversions inside the loop, but with our proposed new value which starts between ArcCos[{1-eps,1}] rather than exactly at |0,0> *)
     thetaListNew = ReplacePart[thetaList, j → thetaNew];
     phiListNew = ReplacePart[phiList, j → phiNew];

     u = RandomReal[{0, 1}];
     α = Abs[f[t, numTimeSteps, mi, mf, thetaListNew, phiListNew]] / Abs[f[t, numTimeSteps, mi, mf, thetaList, phiList ]];

     accept[] := (thetaList = thetaListNew; phiList = phiListNew;);
     reject[] := (Unevaluated[Sequence[]]);
     If[u < α, accept[], reject[]];
     (*Calculate acceptance value α and decide*)

    ), {j, numTimeSteps}];

   {f[t, numTimeSteps, mi, mf, thetaList, phiList], Cos[thetaList], phiList}

   , numSamples]);
```

Figure 2: Here, we create a random pre-Omega list, create Omegas from this list, re-extract theta and phi values we need to put into our `f`. Then, we loop over the actions: a) create a pair of theta and phi near the north pole, b) rotate these by the value at the current index of the lists we just made and make a new Omega from this to re-extract theta and phi new, c) replace the value in the old lists with these at the current index, d) calculate the acceptance ratio by Equation (8). If our random u is less than or equal to α, then accept the change, otherwise no change (Unevaluated Sequence). Finally, we return the final `f` after all Omegas have been considered for change.

Finding c_m in the Monte Carlo Markov Chains case is slightly different although it can still be thought of as doing an integral over the sphere. Except for now we can introduce a probability distribution to bias the paths traveled that are most likely. Our probability distribution is proportional to the absolute value of `f` and it is important to note the role that this probability plays in simulating qubits, especially for future developments of this software.

The goal is to sample $c_m = \int D\Omega f_m(\Omega)$ more efficiently, where $D\Omega =$

$d\Omega_1 d\Omega_2 \cdots d\Omega_N$. Consider a probability distribution proportional to the magnitude of f_m: $p_m(\Omega) = \mathcal{N}|f_m(\Omega)|$. We can rewrite the coefficient as

$$c_m = \frac{1}{\mathcal{N}} \int D\Omega p_m(\Omega) \frac{f_m(\Omega)}{|f_m(\Omega)|}.$$

Note that $\mathcal{N} = p_m/|f_m|$ is a constant, so

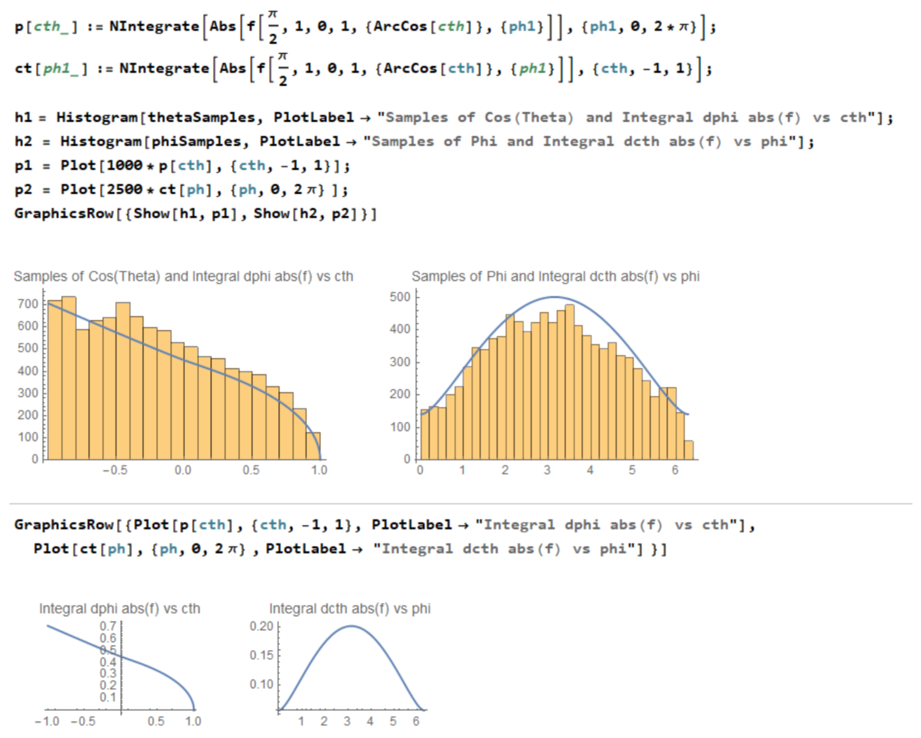
$$\mathcal{N} = \frac{p_m(\Omega)}{|f_m(\Omega)|} = \frac{\int D\Omega \, [p_m/|f_m|]}{\int D\Omega \, [1]} = \frac{1}{(4\pi)^N} \int D\Omega \left[\frac{p_m(\Omega)}{|f_m(\Omega)|} \right].$$

Plugging this into the formula for c_m and noting that the expectation value of a function $g(\Omega)$ over probability distribution $p(\Omega)$ is given by $\int D\Omega p(\Omega) g(\Omega)$, we see that

$$c_m = \frac{(4\pi)^N \int D\Omega p_m(\Omega) \frac{f_m(\Omega)}{|f_m(\Omega)|}}{\int D\Omega \frac{p_m(\Omega)}{|f_m(\Omega)|}} = (4\pi)^N \frac{E[f_m/|f_m|]}{E[1/|f_m|]}. \tag{14}$$

Of which, $p_m(\Omega)$ can be efficiently sampled using Monte Carlo.

We aim to compare the values of the resulting estimates between our first Monte Carlo and this Markov Chain method. First, we must verify that our `Sampler` is working properly by comparing the expected distribution of Cos(theta) values and phi values. Again, we must use Cos(theta) to ensure we are sampling evenly across all states. Below, we plot the integral of the absolute value of f with respect to the two variables and place them on top of each corresponding histogram to compare.



```
p[cth_] := NIntegrate[Abs[f[π/2, 1, 0, 1, {ArcCos[cth]}, {ph1}]], {ph1, 0, 2*π}];
ct[ph1_] := NIntegrate[Abs[f[π/2, 1, 0, 1, {ArcCos[cth]}, {ph1}]], {cth, -1, 1}];

h1 = Histogram[thetaSamples, PlotLabel → "Samples of Cos(Theta) and Integral dphi abs(f) vs cth"];
h2 = Histogram[phiSamples, PlotLabel → "Samples of Phi and Integral dcth abs(f) vs phi"];
p1 = Plot[1000*p[cth], {cth, -1, 1}];
p2 = Plot[2500*ct[ph], {ph, 0, 2π} ];
GraphicsRow[{Show[h1, p1], Show[h2, p2]}]
```

Samples of Cos(Theta) and Integral dphi abs(f) vs cth
Samples of Phi and Integral dcth abs(f) vs phi

```
GraphicsRow[{Plot[p[cth], {cth, -1, 1}, PlotLabel → "Integral dphi abs(f) vs cth"],
  Plot[ct[ph], {ph, 0, 2π} , PlotLabel → "Integral dcth abs(f) vs phi"] }]
```

Figure 3: Samples of the final values of Cos(theta) and phi decided by our Metropolis-Hastings Sampler function at 1 time step with the expected function on top of it

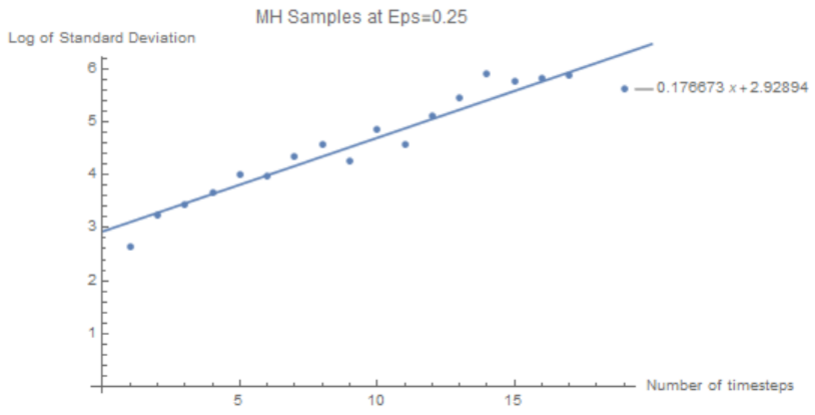
Now that we have shown similarity in these graphs, we can begin plotting the standard deviations of our samples for each value of N as we did before in the Monte Carlo case in order to compare the results. Now, we have one more variable to take into account so we will make three graphs for different epsilon values 0.25, 0.5, and 2. To find the standard deviations values we used the same E we had defined in the Monte Carlo case, except this time, we created a small piece of statistical code in python for finding a fractional error to contribute to our s, which is typically the standard deviation of our samples divided by the the square root of the number of samples. In this case,

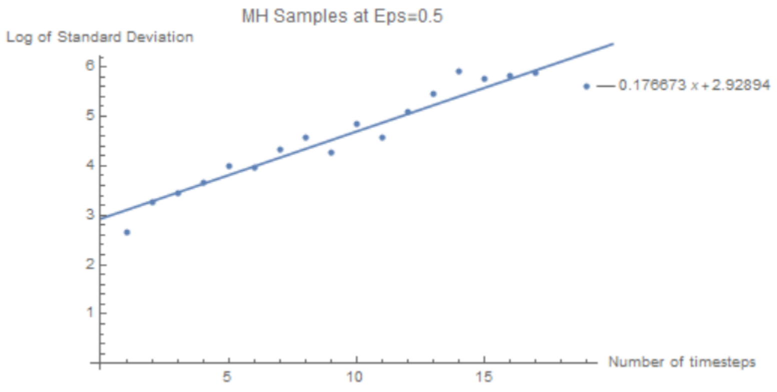
$$s = error_{fractional} \, c_{mean} \tag{15}$$

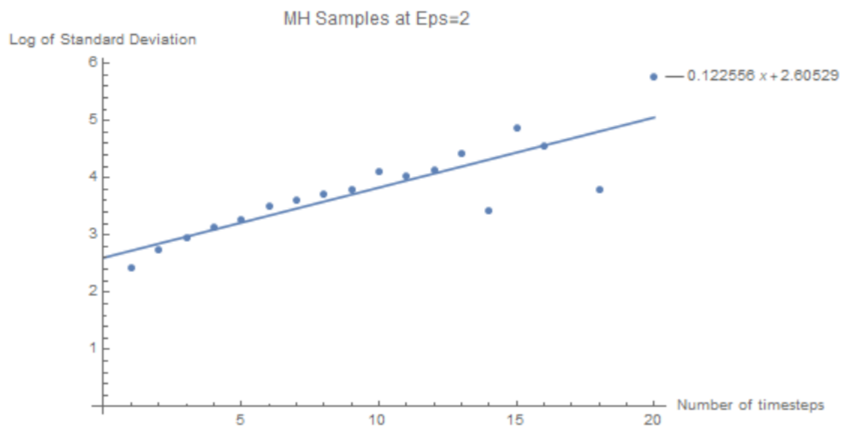
$$error_{fractional} = \sqrt{\frac{2\ error_{numdata}}{mean_{numdata}} * \frac{2\ error_{dendata}}{mean_{dendata}}} \tag{16}$$

$$E = s\ \sqrt{n_{samples}} \tag{17}$$

Here, numdata and dendata are samples of the numerator and denominator of c_m described in equation (12) separated out. This is extracted from our `Sampler` data and is as simple as dividing the smapled `fs` by the absolute value of itself. The error is calculated in a wieighted linear kappa method used in statistics. With all of this, we can graph $log(E)$ vs N for each of the sampled epsilon values.







Comparing our slopes to the slope fitted by the Monte Carlo method, we can see that our slopes are much smaller for these graphs $\approx \frac{1}{2}$ of what it was before. This indicates a smaller rate of change in the standard deviation between timestep values. This shows that the data is more consistently close to the mean than with our other method which has more varying closeness to the mean over the number of timesteps used. Where the mean is our ideal value of c_m.

5 Results

We showed how to calculate expected values of spin coherent states by integrating all possible paths from an initial $|0\rangle$ to a final $|0\rangle$ or $|1\rangle$. We then showed how to do this with Monte Carlo sampling and how standard deviations of our function change with our number of timesteps. Finally, we showed a Metropolis-Hastings method of extracting the same values and compared the results.

Sources

Kim, Monica and Yen Lee Loh. "Visualizing Spin States Using the Spin Coherent State Representation." American Journal of Physics, vol. 83, no. 1, 2015, p. 30., doi:10.1119/1.4898595.