Energy Density Fluctuations in fluxtubes

Rachel Steinhorst

August 27, 2020

Summary

The program edens.cc takes as input Amax (the number of gluons), and Ntraj. Given Amax, it calculates all possible random walks of length $\leq$ Amax (CalcPQCount). By this I mean that for each gluon added, a step is taken in the 2-D space representing the possible eigenvalues (p,q) in SU(3), and so the "path" in pq-coordinate space can be seen as a random walk. See the "papers" directory of this project for a description of addition in SU(3). In this case the flux tube in its entirety must form a color-singlet, so when producing trajectories we will only select paths that start and end at (0,0). For each of a set of values of y, it does the following:

1. Generate Ntraj*10 trajectories by selecting each step weighted by its degeneracy given that the walk must return to (0,0) (FindTrajectory). Each trajectory is also associated with a random placement of gluons in rapidity space between -7 and 7.

2. For each random walk, calculate the quadratic Casimir at each step and use this to find the energy density:

$$\begin{aligned}\epsilon = \frac{dE}{dy} &= \frac{1}{\sqrt{2\pi}\sigma}\int_{-\infty}^{\infty} d\eta \frac{dE}{d\eta} e^{-(\eta-y)^2/2\sigma^2} \\ &= \frac{1}{2}\sum_a \frac{dE}{d\eta}\Big|_{\eta=\eta(a)} \left[\text{erf}\left(\frac{\eta(a+1)-y}{\sqrt{2}\sigma}\right) - \text{erf}\left(\frac{\eta(a)-y}{\sqrt{2}\sigma}\right)\right].\end{aligned}$$

   Here σ is the thermal smearing width. We assume that

$$\frac{dE}{d\eta} \propto C(a),$$

   therefore in this program we take the constant of proportionality to be 1. All moments are then off by a scalar multiple.

3. Average over all trajectories (which are split into 10 samples to estimate error) to find $\langle \epsilon^n \rangle$, and therefore the cumulant ratios $\omega, S\sigma, K\sigma^2$.

4. Write the ratios to an output file (moments.dat), which can be used with moments.py, ratios.py, and altratios.py to graph cumulants and their ratios over y.

The program edens_vsA.cc is similar, but takes y as an input and lets Amax vary. These results are written to moments_vsA.dat and can be used with moments_vsA.py.

In edist.cc, the same procedure for producing trajectories is used, but instead of calculating statistical moments it directly bins trajectories by energy density at mid-rapidity. It takes as input the number of gluons Amax, the number of trajectories Ntraj, and n, which as a default should be 0. You can graph this distribution using dist.py. The option is also available in edist.cc to calculate the distribution of average energy density over n flux tubes, which you can use to see that the distribution becomes Gaussian as predicted by the Central Limit Theorem.

Some Thoughts

Based on results shown in the figs directory, it appears that the cumulants of energy density fluctuations (or at least for $N = 1, 2$) follow the relation

$$C_N = k_N A^N (y^2 - y_0^2)^N$$

where k_N are constants and y_0 is the maximum/minimum value of rapidity in the flux tube, in this case ± 7. Certainly $C_N \propto A^N$, though the relation to y is a little trickier for higher-order ($N = 3, 4$) cumulants due to larger error as N increases and me being too lazy to run at significantly higher Ntraj. Note that in a Poisson or Gaussian distribution, $C_2 \propto C_1$, whereas here $C_2 \propto C_1^2$.

It also seems that the energy density distribution itself is given by

$$P(\epsilon) = \frac{1}{24r^5}\epsilon^4 e^{-\epsilon/r}.$$

Assuming this distribution we find that

$$r = \frac{C_1}{5} = \frac{C_2}{C_1}.$$

This distribution also satisfies the observation from the cumulant ratio figures that $C_N \propto C_1^N$:

$$\begin{aligned} C_2 &= 5r^2 = \frac{C_1^2}{5} \\ C_3 &= 10r^3 = \frac{2}{25}C_1^3 \\ C_4 &= 30r^4 = \frac{6}{125}C_1^4. \end{aligned}$$

Additional fluctuation will also be introduced from fluctuations in flux tube length (by length, here I'm referring to the number of gluons). This could be incorporated by generating the number of gluons for each trajectory according to a Gaussian centered at some average $\overline{A}$. The trick to this is that you'd need to calculate PQCount for the maximum value of A, and the larger this value is, the slower the code is going to run while generating all trajectories. Alternately, you could calculate PQCount individually for each trajectory, but this could potentially slow calculation just as much, I'm not sure. Just make sure you call ClearPQCount each time you generate a new one to avoid costly memory leaks.