DSC Tiling Plan

Part 1: Get the simulations predicting the theoretical blinking patterns.

1. Construct code to go from a DSC vector to a homogeneous psuedostacked bilayer lattice suitable for inclusion in a simulation.
2. Make adjustments to the simulation code so that we can run accurate single-point diffraction simulations.
   1. I’m not certain what this will look like. Maybe I can just pass in a really small unit cell and it will be fine, even with the large-ish beam that Colin seems to have set up.
   2. If not, make an excessively large unit cell and plop the probe right in the center of it to get single DP.
3. Construct driver script for doing a hexagonal raster on the DSC unit cell.
   1. I should be able to work out the boundaries of this unit cell analytically, but I think they give the basis vectors in the Scientific Reports paper (see the Correspondence document and my email to Maddie).
   2. The spacing of the lattice will vary depending on the Sigma ratio between the Moire volume and the graphene volume. If this ratio is large (small Moiré twist angle), then the DSC lattice vectors should be smaller. However, the absolute size of the DSC hexagonal unit cell should remain constant, as it should be determined not by the Moiré twist, but by the primitive lattice.
   3. What I really want then is these constant delimiters, because I am interested in a generic characterization that will work for multiple Moiré twist angles, so the discretization of the DSC cell will be changing.
   4. Therefore, DSC points for doing a simulation should be recorded in fractions of the cell invariant basis vectors, such that -1 < a+b <= 1. (This actually traces out a circle rather than a hexagon, annoyingly.)
   5. Define a sub-lattice where I take some fractional mutiples of a and b to be the tiling basis. (Similar to the Moiré basis that would arise on the DSC cell, but this is arbitrary for the discretization of the simulation.)
   6. Determine all of the hexagonal tile points lying within the DSC cell on the basis of this discretization. These are the hexagonal raster points.
   7. Will need to cut some sub-cells in half on the boundary, for instance.
4. Run simulations for the DSC hexagonal raster unit cell
   1. These simulations themselves should be fairly fast, as it’s a point calculation for each hexagonal pixel and we can interpolate at the end anyways. I anticipate that we can get it done in a couple hours with maybe a 20x20 = 400 subpoint tiling of the DSC cell, for instance.
5. Select an ROI for each graphene peak of interest. Obtain integrated intensities from the center of the disk as the DSC position is hexagonally rastered to quantify blinking.
   1. Perhaps use my hexagonal lattice mask function, with the fixed point in the center of the image, then choose a very small radius.
   2. Each simulation is now defined by twelve numbers giving the blinking intensity.
      1. These numbers should be relative to maximum intensity of the integration region, so the twelve numbers are in the range [0,12].
   3. Will need to do some validation work to make sure that the ordering of the windows is correct.
   4. Make plots, in two ways. (Both will require custom code.)
      1. Line plots of the intensity. Probably do this in the same way they do band structure, along the three high symmetry directions of the hexagonal cell. There will be six lines, (or twelve if we include the second ring of graphenes), which we can color-code and have a reference image of a hexagon next to it. (Color-coding will be challenging – May need to do one plot which is all B&W, and couple other subplots which are color-coded in six colors for each of the two graphene rings.
      2. Psuedocolor plots. Here develop the custom hexagonal plotting code discussed in the Correspondence document. For each peak, show how the intensity goes up or down.

Part 2: Validate the retrieval method using the large simulation of 4DSTEM data that I ran recently (full 2D raster).

* Use the same fitting method as below for the experimental data, but because the simulation is based on true TBLG Moiré lattice while the fit is based on hypothetical psuedostacking regions, this will tell us how well the psuedostacking approximation works when going through the diffraction patterns.
* We can compare directly to the psuedostacking assignments from the Moiré DSC, used directly rather than inferred from the fit to the stacking patterns. This will tell us how well psuedostacking regions are preserved when going through the diffraction patterns and the interpolation. I think they should be well preserved.

Part 3: Fit the simulation blinking theory to the observed 4DSTEM data:

1. Set hexagonal ROIs on the lattice of the experimental data. Obtain the integrated intensity for a small region at the center of the disk, best representative of blinking.
   1. Each diffraction pattern is now characterized by a set of twelve (or maybe more, depending on what is visible) numbers that quantify whether the graphene disks have blinked or not.
   2. The twelve numbers should again be relative to the maximum intensity on the disk, so between 0 and 1. This way simulation and experiment will be on the same scale for fitting.
2. Build functions that will return a interpolation of the simulation data as a continuous function of the DSC vector valued input.
   1. This could be a spline to be most consistent, or even mostly simply, a simple linear interpolation between the nearest neighbors.
   2. Note! This is a bit tricky because it is a 2D interpolation with reference data that is not square, but in a hexagonal pattern. Will have to be careful to make sure this works out correctly.
3. Fit each experimental diffraction pattern with a simulated diffraction pattern.
   1. The fit will take place on a diffraction-pattern-by-diffraction-pattern basis.
   2. This will be a nonlinear optimization in two parameters, which are the two coordinates of the DSC vector. It would be best to explicitly construct bounds on the search space, which should be possible because a regular hexagon is convex, but it might require a little work. A system of inequalities for fmincon might do the trick – would have to see. If not, can inscribe the hexagon in a square, bound with the square, always start the optimization within the hexagon, and then do a periodic wrap around if it should leak out any. After the function terminates, convert the DSC to a reduced representation with a modding function.
      1. Actually you could do this without any bounds at all, and that might work just as well. All conversions would be internal to the function.
   3. The objective function will be the RMSR between the twelve numbers characterizing the simulation (generated at the particular DSC guess) and the twelve numbers characterizing the experiment used at the moment.
   4. This should be weighted least-squares, where the weights come from the maximum intensity of the disk when the blink is fully on (in the experimental diffraction pattern).
      1. The basic idea would be to take the maximum of the maximum disk intensities, and assign that disk a weight of 1. Then, reference all other disks to it for fractional weights.
      2. But depending on how much the center beam is polluting the first ring of graphenes, this might not be ideal.
      3. We could also use a Poisson MLE estimator.
      4. Or, perhaps best, do the relative weighting idea, but have it weighted by the difference in the height of the disk and the height of the background.
4. Visualize the result
   1. For each scan position, we will have obtained an arrow from the optimization that constitutes the two-component DSC vector. Can therefore do a quiver plot in 2D Cartesian showing the DSC vector field. This preserves DSC direction.
   2. Alternatively (and this is the plot that is more likely to look cool), assign psuedostacking regions via arbitrary cutoffs in the DSC lattice. Set perhaps a true AA, nearly AA, true AB, nearly AB, true SP, nearly SP, etc, so that there is a little bit more information than just the three color map. (Or that could work too). This is a discrete colormap.
   3. The final possibility is to do a continuous colormap, where we are measuring distance from AA, AB, and SP simultaneously. Most naturally, these can be R, G, and B, respectively. (Perhaps red, blue, and grey could also work, or some other color choices so as to be okay for colorblind people.)