10/21/18 Experimenting with my 1D python DD code:

Testing speedups due to different modifications:

Running code with same parameters as the matlab and c++ codes and -0.5 to 1.2V and only outputting the JV data and no excessive output to terminal.

Original time: 143sec

1. Adding @jit (from numba) to Thomas\_solve

Sped up to 103sec

**Note: the fast version of numba, with the nopython=true tag, raises exception meaning my function can’t be optimized that way! Same goes for all other functions tried to add @jit with nopython tag, it gave an error.**

2. Try adding @jit to other functions, i.e. continuity setup, poisson setup, Bernoulli… With all of these, the cpu speed actually slowed, to ~120sec.. So I guess the time to compile is taking longer than before..

So removed @jit from these functions.

3. checked if in Thomas\_solve, when using @jit, directly accessing the diagonal arrays through system. (the object) makes a time difference:

**Yes, large slowdown!, to 152 sec from 103 sec!**

So it is better to make a copy (i.e. I was using deepcopy) of the arrays and then use those…

**Note: if access the arrays directly using the objects with numba, it is actually slower than without adding the @jit tag! (w/o @jit was 138 sec).**

**4.** Check if the np.copy() is faster than using copy.deepcopy for within Thomas solver:

This is for copying the diagonal, upper, lower, and rhs arrays.

Same time: 103sec 🡪 the copying seems not the bottleneck…

5. Check if passing explicitely these arrays to the Thomas\_solve function, instead of passing an object is faster.

**But I still need to do the copy for the diagonal, otherwise it will be modified by the thonmas solve! Rhs is ok to be changed, b/c it is resetup in every iter anyway…**

**Yes, that is faster!!: 96.6 sec🡪 makes sense that passing less stuff (i.e. not the entire object) is faster!**