Cythonizing Code **KAM 5/12/2020**

This document describes the procedure used to develop the Cython (Python + C) Code that was compiled into a module used for studies with WrightSim. URLs to helpful links are at bottom.

Software. I used a PC with Win 10 64-bit with the following software installed, and validated a Linux variant of installation with a version of Ubuntu. PC:

Anaconda3

VSCode with Git -- installed under Anaconda3

mingw-w64 -- http://mingw-w64.org/ -- a gcc-like C compiler for Cythonized code with Windows 10 builds. This is not installed under Anaconda, nor could I find a means of doing so quickly.

python 3.7.6 -- installed on Anaconda3 prompt using `conda install`.

numpy, scipy -- similar to python

ffmpeg, celluloid -- `pip install` moviemaking packages for studying simulations

Cython -- installed using `pip install` under anaconda3

WrightSim, WrightTools (via Gitlab repository, not via conda)

I also used Visual Studio (2015) to examine the .c code generated by Cython. This is an unnecessary step.

Skip the next paragraphs and proceed to the numbered list below if you just want to see the software installation steps rather than how the .c code was developed.

Procedure. Using VSCode I developed the Cython file 'npropagateS15.pyx' according to Cython instructions. The Cython pyx file looks primarily Python like for this study. Arrays used internally are defined as "C-like" with the **cdef** statement, and are then accessed with C-like instructions (e.g., "[][]" rather than "[,]". The static-size of slices of Hamiltonians, always being 9 x 9 , are well-suited for declaration as such in cython. Streamlining code was possible through the use of so called "memoryviews" of the input functions, and knowledge *a priori* of the shape of the 3D Hamiltonian input allowed for more efficient coding. Thus, static arrays needed are simpler 9-element vectors used in density vector propagation, as well as one 2D array that carries the density vector elements along a final time *t*. The 2D array was declared C-like. Cython did not allow C-type dynamic array allocation readily except via memoryviews, so a dummy "memoryview" of the expected output 2D array was incorporated

into the function call for easy copy into a C-type array. The output array was a numpy instanced array that simply copied the internal 2D array data. Because `complex` matrix multiplication involves crossterms of the real and imaginary parts, the original 3D Hamiltonian (and associated density vector) was split into real and imaginary parts *before* the function call, as it was thought Numpy arrays had these already set and it would make sense to utilize the Numpy values as best as possible prior to the function. The output numpy array was of type complex so as to pass data back in just one object and as close to the original code as possible.

The remainder of the code is straightforward and can be compared stepwise to the original propagate.py script, recalling that the procedure shown is in effect a 2nd-order Runge-Kutta matrix-vector propagation procedure. Of key importance is the exclusion of all non-essential Hamiltonian matrix elements. This reduces the number of multiplication/summation steps from 486 (x 2 for 2nd-order) to 138 (x 2). An extremely thorough search of numpy and CBLAS libraries was made. At no instance in the normal matrix ("dot") -vector product libraries was there a provision for discounting a multiplication step if the matrix element was zero, the reasons for which are not elaborated here.

Cython directives can be employed to skip array bound checking and other checks to speed code. These were not found to speed code greatly, though the timing checks were not careful in checking the cython module and more studies with direct timings of this function can be performed for further optimization with these above directives. See below for links on how this can be accomplished and the other document on further elaboration as to the above statement on timing.

After code was developed, the following steps were run via shell prompts.

1. From Anaconda prompt: `cython -3 npropagateS15.pyx` .This should create a corresponding, very large *.c* file in the same directory.
2. From mingw-w64 prompt and in npropagate's directory:
```
gcc -shared -pthread -fPIC -fwrapv -O3 -Wall -fno-strict-aliasing -DMS_WIN64 -isystem C:\ProgramData\Anaconda3\pkgs\python-3.7.6-h60c2a47_2\include -s npropagateS15.c -LC:\ProgramData\Anaconda3\libs -lpython37 -o npropagateS15.pyd
```

(The include folder may be different depending on the Conda installation. The `-DMS_WIN64` is necessary to allow interpretation of certain pointer code in the .c file. "O3" is the optimization level for this code as it was found to provide fastest timings. The shared tag and libs link is necessary for the shared object pyd creation. `-fPIC -fwrapv -fno-strict-aliasing` did not seem to affect results but were left in from other website recommendations.)

3. Copy-paste of the resulting pyd file into the WrightSim build or script directory containing propagate.py.
4. Use the alternate form of propagate.py found in this repository for testing of the pyd file. (Drop and replace into your working WrightSim folder.)

NOTES: There are no mechanisms to handle exceptions...see "Gotchas" link below for more on to how to make this cythonized code safer. Inputs are presumed of type np.float64 which are double precision. Thus the original Hamiltonian must be of type complex128. Failures may result if these datatypes are not used as inputs. Memory leaks have not been tested, though the code indicates it is unlikely to occur there and more likely (if ever) in the Cython-based .c file.

LINUX INSTALL

Cython and other modules can be installed from the shell using pip install. If Python was installed via Conda, the following gcc command can be used instead:

```
gcc -shared -pthread -fPIC -fwrapv -O3 -Wall -fno-strict-
aliasing -I /home/<username>/.conda/pkgs/python-3.7.1-
hd21baee_1001/include/python3.7m -s npropagateS15.c -L
~/home/<username>/.conda/pkgs/python-3.7.1-hd21baee_1001/lib
-o npropagateS15.pyd
```

Links used

https://cython.readthedocs.io/en/latest/src/userguide/numpy_tutorial.html#numpy-tutorial

http://docs.cython.org/en/latest/src/userguide/memoryviews.html#cython-arrays

https://stackoverflow.com/questions/11182765/how-can-i-build-my-c-extensions-with-mingw-w64-in-python

http://docs.cython.org/en/latest/src/userguide/source_files_and_compilation.html#distributing-cython-modules

https://suzyahyah.github.io/cython/programming/2018/12/01/Gotchas-in-Cython.html

https://medium.com/@stefanobosisio1/cython-fast-as-gpus-without-gpus-2c2e52cc4c42