COSMOS 2022, Computational Lab 11

[01] On CANVAS you will find the two files

```
eigenvaluesopenblas1dchain.c
get_eigenvaluesv2.h
```

needed for all our work this week. eigenvaluesopenblas1dchain.c is a C code which
(1) sets up the Hamiltonian matrix (named mat) for an electron 'hopping' on a one dimensional chain of N nucleii, and then prints mat to the screen.
(2) Diagonalizes mat.
(3) Prints the eigenvalues (real part named val_r and imaginary part named val_i) and eigenvectors (named vecs) to the screen.

The 'hopping' energy is set by the input 'thop' and the energy to reside on a nucleus is the input 'E'. Physically thop is determined by the overlap between the wavefunctions of electrons on neighboring nucleii. If the nucleii are far apart thop is small. If the nucleii are close thop is large.

eigenvaluesopenblas1dchain.c calls an an eigenvalue/eigenvector solver which is contained in the second file get_eigenvaluesv2.h. This uses an 'open' version of "BLAS" (Basic Linear Algebra Solvers), the most widely used package for doing numerical linear algebra. You can think about it as like 'math.h' and 'stdio.h' which when included in your past C codes allowed you to do math operations and input output. This header file allows you to diagonalize matrices.

To use the code you will need to install openblas on your computer. Do this with:

```
sudo apt install libopenblas-dev
```

Then compile it with:

```
gcc -o eigenvaluesopenblas1dchain.e eigenvaluesopenblas1dchain.c -lopenblas
```

Check your code for $E = 5.3$ and thop$= 0.8$. You should get:

```
Eigenvalues:
    3.796492      0.000000
    4.074329      0.000000
    4.500000      0.000000
    5.022163      0.000000
    5.577837      0.000000
    6.100000      0.000000
    6.525671      0.000000
    6.803508      0.000000
```

The first column is the real part of the eigenvalue. The second part is the imaginary part. All the eigenvalues here are real because our matrix is real and symmetric. As we have discussed,

in all of quantum physics the matrices corresponding to measurable quantities will have real eigenvalues.

We'll talk more about these codes in class, but let's summarize key points here.

The first comment is 'philosophical'. In physics (and science generally), there are two broad categories of theoretical work. In the first, one attempts to answer general, qualitative questions: Is a particular solid (geometrical arrangement of atoms) a metal or insulator? Are the electron wave functions spread out or localized? Is the solid likely to be magnetic or not? In the second, the goal is to compute precise, quantitative values: What is the resistance, in Ohms, of a particular solid composed of copper, aluminum, or NaCl? What is the specific magnetic transition temperature, in degrees Kelvin, of a piece of iron?

In the next week or two we will see how we can address the first type of question. In some sense this is the most satisfying type of physics because it involves the understanding of general principles. On the other hand, if you really want build a working photovoltaic array and produce solar power, you have to be able to answer the second type of question.

We'll begin with a simple model of the simplest type of solid: Consider a one dimensional chain of identical atoms. We will characterize it with two parameters:
(a) the energy E for an electron to sit on any nucleus; and
(b) the energy thop for the electron to move ('hop') between adjacent nucleii.

We need to write down the Hamiltonian, a matrix whose eigenvalues give the possible energy levels of an electron living on this one dimensional chain. The way we construct the Hamiltonian is by following these simple rules:
(1) number the N sites from 0 to $N-1$;
(2) put the value E along the diagonal of the matrix;
(3) for any pair of sites (i, j) which are neighbors, put the value thop into the [i][j] entry of the matrix;
(4) all other elements of the matrix are zero.

Take a look at

`eigenvaluesopenblas1dchain.c`

and see if you can see that it is precisely following these rules. You will notice that, after asking for you to input E and thop it first implement step **(4)**, setting all entries in whole matrix (named mat) to zero. Next it implements step **(2)**, and finally step **(3)**. You should look closely at the implementation of step **(3)**. Why are rows 0 and $N-1$ singled out and treated differently from all the rest of the rows which sit inside the loop?!?!

These rules seem to be pulled out of thin air, but we can make them a bit plausible by thinking in the following way. First, I hope it seems reasonable that the ability of an electron to move been two nucleii at i and at j might be associated with the matrix element mat[i][j]. To see this, recall that a matrix converts one vector v into another vector w. If you think about it, mat[i][j] is precisely what controls how much the j-th component of v influences the i-th component of w. So here the mathematics, as so often happens, is mimicing the physics: mat[i][j] encodes the ability of nucleus i to take an electron away from nucleus j.

It is also useful to think about what happens if thop= 0. This represents the case when the electrons cannot move from nucleus to nucleus. Run your code with thop= 0.0 and E=5.3. What does the list of eigenvalues look like? Can you argue why this is the right answer? The way you should look at this is that thop=0 means you have N atoms so far apart that they basically are independent of each other. If they are the same type of atom, the energy of an electron is the same for all of them. This is why you have a bunch of identical eigenvalues.

Now try thop= 0.01, E=5.3 and thop= 0.1, E=5.3. What do you observe happening? The identical eigenvalues are becoming different, spreading apart. This is the basic picture of the formation of an energy band in a solid. The atoms now 'touch' each other and affect each others' energies.

Whenever someone gives you a code and tells you it does something you are well advised to check it. Before doing more physics, let's convince ourselves the eigensolver works. To do this, modify the code as follows:

```
#include <stdlib.h>
#include <stdio.h>
#include "get_eigenvaluesv2.h"
int main(){
int i,j,N=2;
double mat[N][N];

mat[0][0]=3.8;
mat[0][1]=0.6;
mat[1][0]=0.6;
mat[1][1]=2.2;

double val_r[N],val_i[N],vecs[N][N];
get_eigenvalues(N, mat, val_r, val_i, vecs,1);

printf("\nEigenvalues: \n");
for (i=0; i<N;i++){
printf("%12.6lf %12.6lf\n",val_r[i],val_i[i]);
}
printf("\nEigenVectors: \n");
for (j=0; j<N;j++){
   for (i=0; i<N;i++){
      printf("%8.4lf ",vecs[i][j]);
   }
   printf("\n");
}
return 0;
}
```

You might recognize the matrix as the one from the notes that has eigenvalues $\lambda = 2, 4$.

Check to see the code works. Are the eigenvectors also correct? Check out pages 11 and 12 of the notes on CANVAS. You will need to normalize the eigenvectors, since that is what BLAS is doing.

[02] Diagonalize the matrix

$$M = \begin{bmatrix} 7 & 1 \\ 1 & 7 \end{bmatrix}$$

"by hand".

[03] Now use your code (on page 3) and verify you get the same answers.

[04] (doing by hand is a bit tricky...) Diagonalize the matrix

$$M = \begin{bmatrix} 2 & 0 & 0 \\ 0 & 7 & 1 \\ 0 & 1 & 7 \end{bmatrix}$$

"by hand". Now modify your code from page 3 (do you see how to do this?) and verify you get the same answers. If you cannot get the answers 'by hand', use your code anyway and see what it says. The answer might help you figure out the 'by hand' solution, especially if you use the results from **[02]**.