Illustrations for How to use the codes

The three pieces of codes here are program that uses CPU to calculate the diffraction pattern of an arbitrary sample. Let's have a look at each of them and learn how to use them. To have a better understanding, I suggest to have a copy of the related paper at hand. The paper can be downloaded at https://arxiv.org/abs/2010.06126. I will use "our paper" to refer to it in later paragraphs.

I will explain each piece of the codes here. The three pieces of codes are independent, you only need to read related sections below to learn about how to use them. However, I strongly suggest you read the LAST section before starting running any of them.

Section 1

Let's first look at "whitexray.cpp". This piece of code is based on Laue method, details has been introduced in our paper, section 6.1 & 5.1. After finished compiling the code, you also need a sample file has the name "config.xyz" at same directory as the executable file. Examples of .xyz files are available at https://github.com/statisticalmechanics/scatter/tree/master/xyz, and for the purpose of testing this code, I would like to suggest that you download the file "test_sample.xyz" under the .xyz folder, and rename it to be "config.xyz" and put it in the same folder as the executable.
Then we can start to run the code. You will be asked to input some parameters. Let's look at them respectively:

1. Front (0) of Back (1)_
   This asks whether you want to use transmission method or back-reflection method, which is described at FIG. 6 in our paper. Input 0 means transmission method and 1 means back-reflection method.
2. Input LOWER limit for wavelength_
   As Laue method uses white x-rays which contains waves in a frequency range, you need to input the lower and upper limit for the range of frequency. Here let's input the lower limit, in the same unit as the coordinates in .xyz file we downloaded before, i.e. in unit of "sigma", the length of atomic diameters.
3. Input UPPER limit for wavelength_
   Same as previous step, we input the upper limit here.
4. Input number of particles_
   Here we need to tell the program how many atoms we have in our sample. Though it is already determined in the config.xyz file, I still leave it as an input-determined value for the reason that you can adjust it freely and to avoid nonstandard .xyz sample files.
5. Input x_Resolution_
   This parameter, along with the next Input y_Resolution_, determines the size of the image you get. However, we should notice that, SIZE OF IMAGE and SIZE OF PATTERN is two different thing, i.e, if you find the diffraction pattern you get looks too small,

increasing this parameter will not help. However, if you find the diffraction pattern you get seems incomplete, and looks like being cut at the edges, then increasing this parameter will help.

6. Input y_Resolution_
   See above.
7. Input distance (D)_
   This parameter is the distance from the sample to the screen, i.e. the D in FIG. 6. We can observe that, this parameter also determines the image result you get. Actually, parameters 5,6,7 determines the pattern size and image size together, and you should pay attention to all of them in order to get an appropriate result.
   Also, it can be seen from FIG. 6 that, the units for the three parameters do not matter. It's the ratio between them that really determines the result. We can assume the units are all meters here, or centimeters, whatever you like.

Now all parameters has been set, we can start to run the code. After the program finished, you will get an output file "Ixy.dat", which contains many lines, each line represent a point on screen, and there are three columns: the first column is the x coordinate on screen, the second is the y coordinate, and the third is the intensity. Yet here I would like to suggest you do not start running this code now, it would be better if you have read the last section before running the code.

Section 2

Now let's go to the code "powder_even.cpp". This piece of code is based on powder method, as introduced in section 8 in our paper. Now let's look the input parameters.

1. Input number of particles, any value bigger than 10000 is illegal_
   It asks you to tell the program how many atoms we have in our sample. Though it is already determined in the config.xyz file, I still leave it as an input-determined value for the reason that you can adjust it freely and to avoid nonstandard .xyz sample files.
2. Input number of points on a sphere_
   This parameter seems confusing. Detailed illustrations can be found in paper, here I would like to offer a brief explanation:
   For powder method, the sample is "polycrystalline powder" containing many tiny crystals, oriented randomly, in all directions. In order to simulate it in program, I didn't really try to generate a sample file of "polycrystalline powder" which contains coordinates of many crystals oriented in every direction. Instead, I found that, having many tiny crystals oriented randomly in space is the same as having many q vectors oriented randomly in space (or it can be understand in this way: having many tiny crystals with different orientations is the same as having incident x-rays in many different directions but keep the single crystal static).
   Therefore, this parameter actually means: how many tiny crystals, which are oriented differently, do you have in your "polycrystalline powder" sample? Or more generally, how fine the powder sample is?
3. Input wavelength_

This parameter is the x-ray wavelength you choose, in unit of "sigma", the length of atomic diameters.

4. Input increment for theta_

   For powder method, the common convention is to have a result figure which describes the intensity over the diffraction angle "2\theta". This parameter is to determine while doing the experiment, how fast should the angle increase, i.e. while "2\theta", the x-axis, goes from 0 degree to some upper value, what is the length for each step?

Now all parameters are set. If you start running, the program should generate an output file, which contains many lines, each line represents a theta value for the x-axis of the result figure. And there are two columns, the first column is the theta value, the second column is the intensity. Yet here I would like to suggest you do not start running this code now, it would be better if you have read the last section before running the code.

Section 3

Now let's look at "cubic_q.cpp". Details about this methods is illustrated in section 7.3. Let's look at the input parameters:

1. Input configurations_

   This asks for the number of configurations, in case you have a sample which is changing with time that contains more than 1 configuration, or you want to calculate the average of samples at different times. If you only have one configuration, then just input 1 here.
2. Input length for x direction_
3. Input length for y direction_
4. Input length for z direction_

   These three parameters are L, the linear dimension of the cubic simulation box, as described in our paper. And Δq is calculated from this value, also illustrated in paper, at section 7.3.
5. Input number of particles, any value bigger than 10000 is illegal_

   It asks you to tell the program how many atoms we have in our sample. Though it is already determined in the config.xyz file, I still leave it as an input-determined value for the reason that you can adjust it freely and to avoid nonstandard .xyz sample files.
6. Input upper bound for x side length of q cubic, any value bigger than %d is illegal_
7. Input upper bound for y side length of q cubic, any value bigger than %d is illegal_
8. Input upper bound for z side length of q cubic, any value bigger than %d is illegal_

   These three parameters are n_x, n_y, n_z, also described in section 7.3.
9. Input resolution for drawing S(q) vs q_

   This is also described in our paper, section 7.3: "When reporting the result of S(q), one can assign q's into bins of equal size or just use the original q values visited by the lattice points." And this parameter "resolution" means "size of bins".

Now all parameters are set. If you start running, the program should generate three output files. The first one. “out.xyz”, contains coordinate of q vectors and related S(q) values. The first three columns are coordinates for q vectors. The second file, “out.q”, contains the same information as the previous one, but only magnitude of q is presented. The third file “out.q.draw”, contains the averaged S(q) values based on similar q values. Yet here I would like to suggest you do not start running this code now, it would be better if you have read the last section before running the code.

Last section

Now we have reached the last section. As I have mentioned many times before, it would be better if we read this section before we start running. Here I want to mention an important issue. If you start running the three pieces of codes we talked above, it might take you hours. The reason is that, to calculate the diffraction pattern of a system, we need to add up contribution from every atom, this is quite a large number. As we loop over all atoms in the system, it requires a very long time to finish.

Then a new idea emerges: why don’t we adopt a parallel computing method? If we can calculate the contribution of each atom parallelly, it will save lots of time. Therefore, I wrote codes based on GPU and OpenCL, which are at https://github.com/statisticalmechanics/scatter/tree/master/code_gpu.

Now you can start to test the three pieces of codes. And during the long time you are waiting for the results, I suggest you to go to the GPU folder and have a look at the codes there. You will find them much more interesting and convenient. If you cannot bear the long boring waiting time, go and try the GPU codes! Generally, they can give the result in seconds.