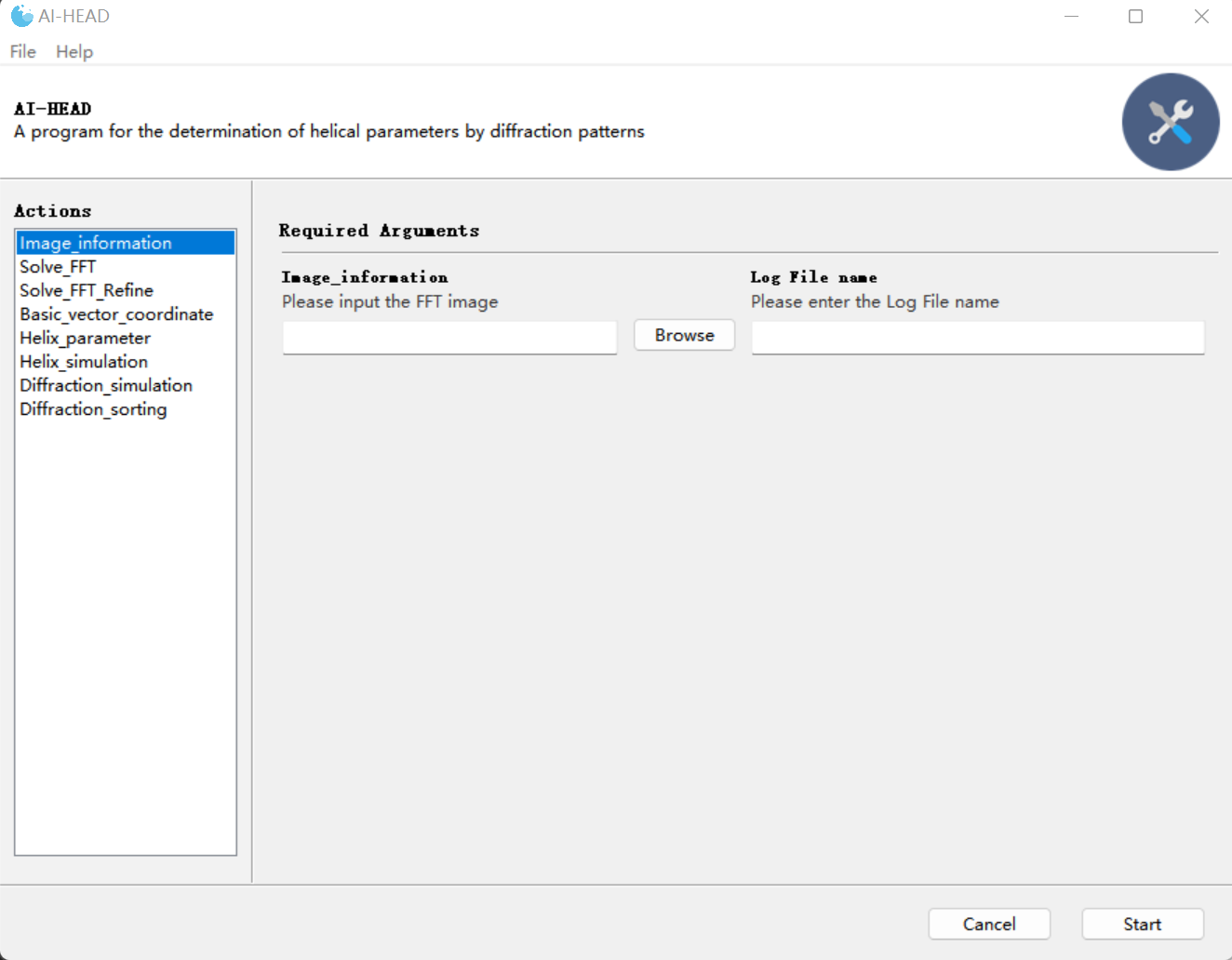
**Auto-Indexing of Helical Diffraction pattern (AI-HEAD) Tutorial**

# A brief introduction of AI-HEAD

AI-HEAD is a program for the determination of helical parameters by diffraction patterns. This program can quickly and automatically conduct the indexing of the diffraction pattern according to the grayscale distribution of the diffraction pattern provided. Researchers can choose an ideal indexing result among various indexing methods and use it to calculate the helical parameters automatically. This would be a very helpful tool for beginners who conduct the helical reconstruction, as you only need to provide a projection of the helix (or a diffraction pattern of this helix) in mrc format.

Moreover, AI-HEAD provides a user-friendly interface, which the user could use various module in turn and simply conduct all of the jobs in it by clicking and entering.



Run the program

Input interface

Module List

The interface of AI-HEAD

More details about this programs could be got at (doi:xxxxx).

# The tutorial of auto-indexing of helical data(projection) in EMDB

## Auto-indexing of diffraction pattern

### Preparing the diffraction pattern of the helix

Now, let’s begin our work. Here's a simple tutorial using the sample data (SIRV2 EMD-6310). First of all, you need to get the diffraction pattern of the helix. You could finish this step by using “Basic\_vector\_coordinate” module in AI-HEAD.

There are 4 inputs in this module:

**-Original micrograph:** You could load your data (in mrc format) in this interface.

\*The data provided here requires equal length and width, 1024 or 2048 is recommended. The longer the helix, the sharper the diffraction layer lines, and the more accurate the indexing results.

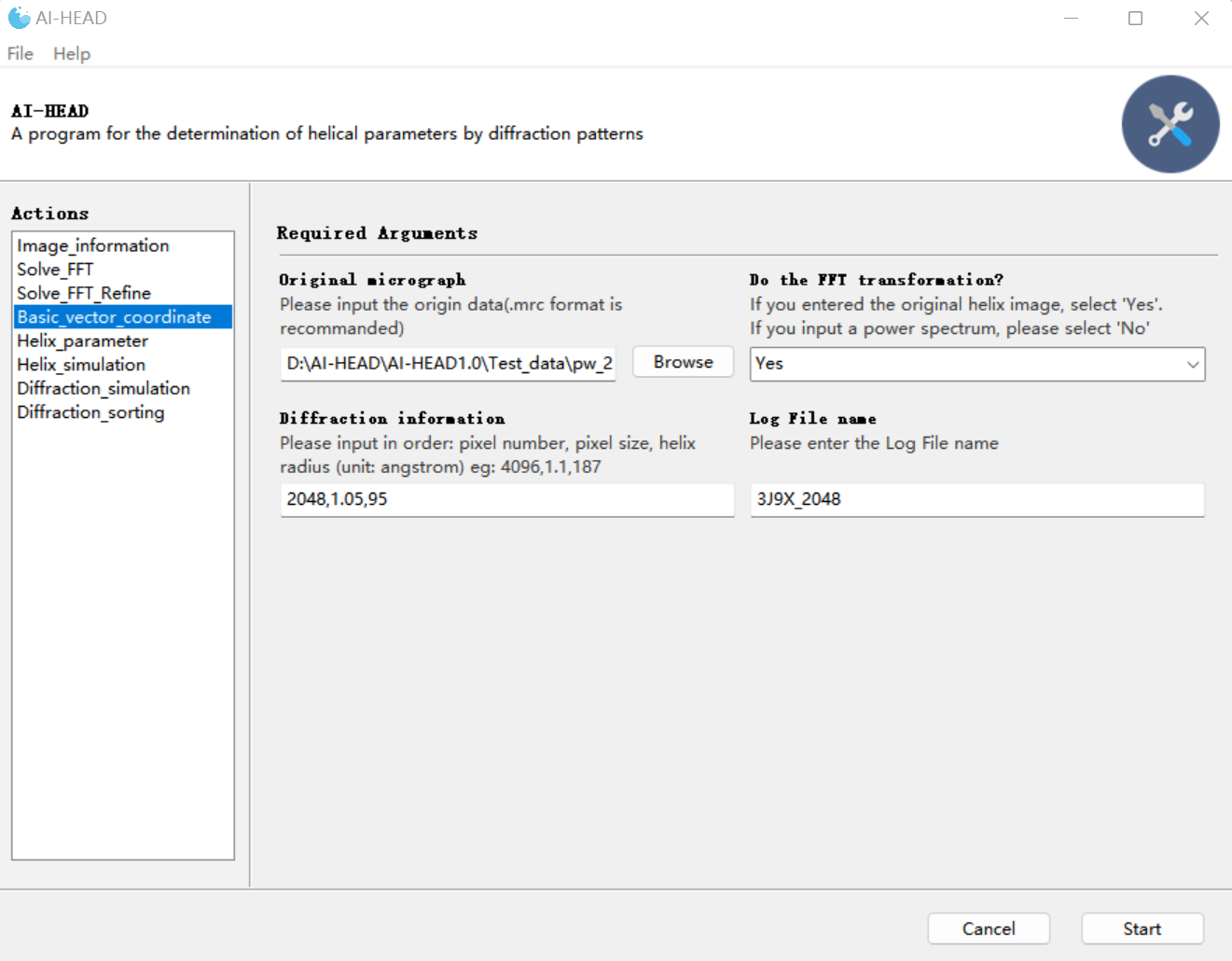
**-FFT opinion:** Since the process of indexing is conducted in the Fourier space. If you provide a projection of the helix, you need to choose “yes” here, and AI-HEAD will do the Fourier transformation automatically; if you provide the power spectrum of the projection of helix, then you just choose “No” here.

**-Diffraction information:** Here you need to tell this program the basic information of your data, which is consist of the size of the data, the pixel size of the data and the radius of the helix (in angstrom).

\*These data are separated by commas.

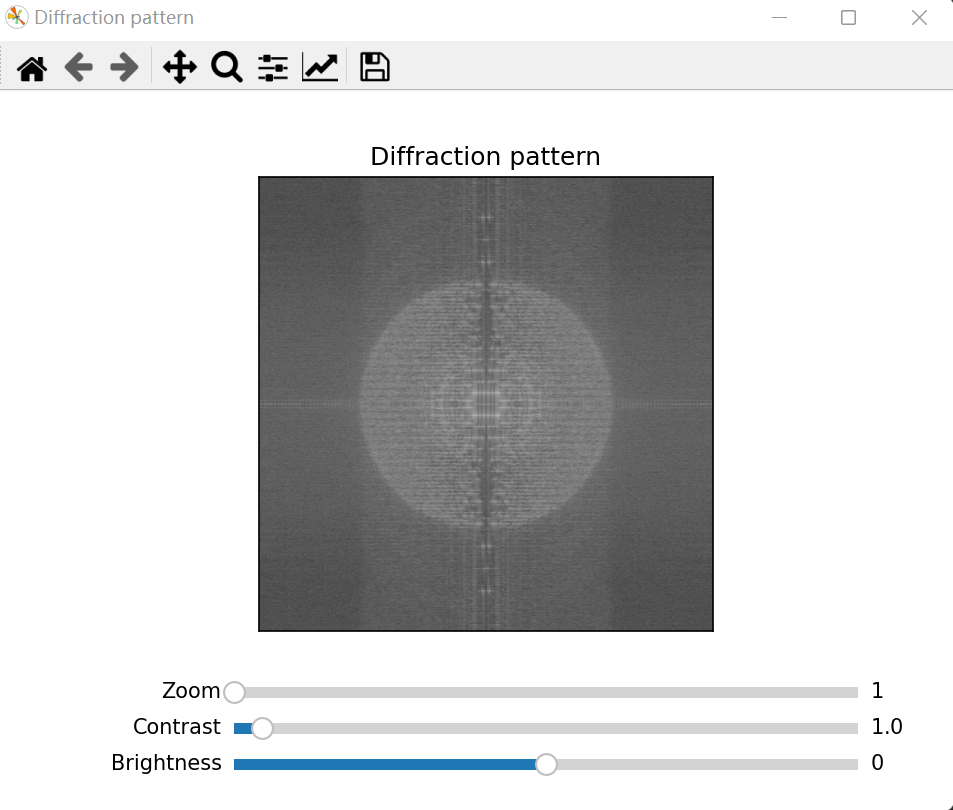
**-Log file name:** This is where you need to enter the name of the log file. Since you need to find the coordinates of the base vector for indexing results here, this step can be done with one click of the interface. The corresponding coordinate information is automatically recorded in the log file.

\*In fact, we just check the diffraction pattern here, so no coordinate information will be generated.



Get the diffraction pattern of the helix

Once we have entered all the information, we can click "Start" to run, and the program will generate an interface of the diffraction pattern:



The power spectrum of the projection

Since the power spectrum obtained by direct Fourier transform can be noisy, and indexing requires only a few major layer lines based on the low frequency region for analysis, the program provides three sliders to adjust the image.

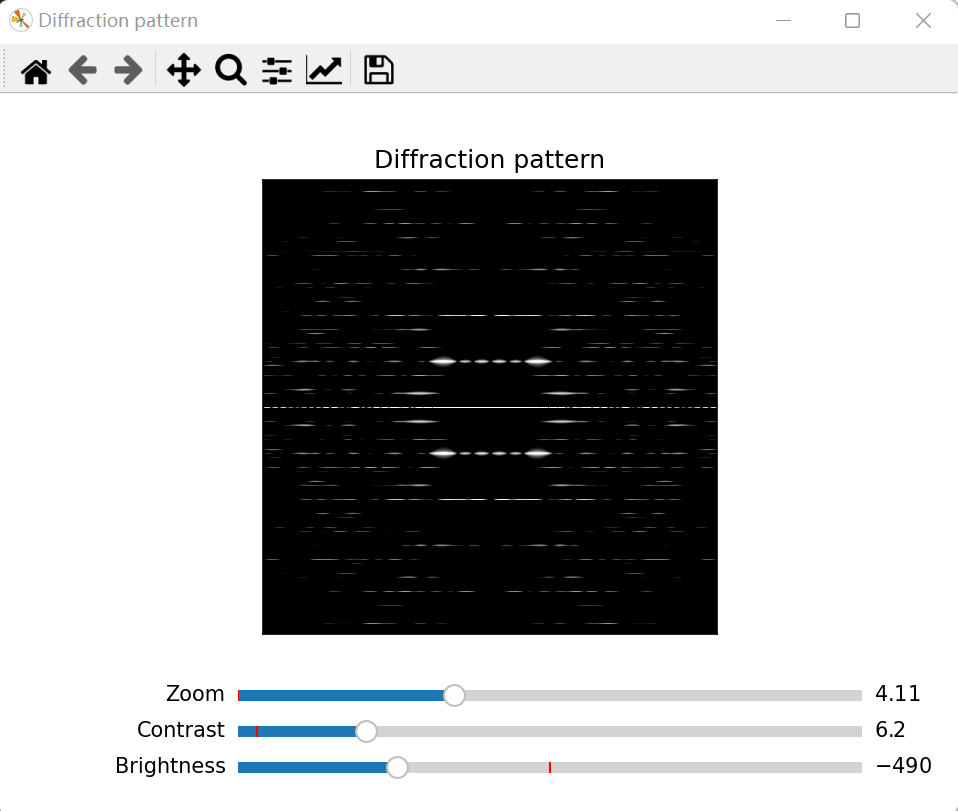
**-Zoom:** This slider can be used to zoom in on the target area, or it can be done by sliding the mouse wheel.

**-Contrast and Brightness:**

The image adjustment calculation formula is as follows:

Image\_adjusted= (Image- mean (Image)) ×Contrast + mean (Image) + Brightness

You could remove excess noise by adjusting the Contrast and Brightness sliders to preserve the main diffraction layer lines. In this case, for example, this step could be done like this:



The diffraction pattern of the projection

And then, you can just press “space”, then the file named “diffraction\_pattern” could be saved for subsequent indexing, **and a “value” file which contains the information of adjustment was saved at the same time.** You could also get the diffraction pattern of the projection by other software such as EMAN2 and so on. However, in order to facilitate the later screening of data, we recommend use AI-HEAD to finish this process.

\*The “value” file is important for the sorting of the data later.



The “value” file

### Get the information of the diffraction pattern

OK, now that we've finished our preparation, let's get back to our indexing. To achieve indexing of diffraction patterns, it is necessary to know the information of the diffraction center of the diffraction pattern and the information of all effective diffraction points. In fact, the program does not know what a "diffraction pattern" is, nor does it know what information to obtain from the "diffraction pattern". So, AI-HEAD needs to get all of the information by the “Image\_information” module.

There are 2 inputs in this module:

**-FFT image:** You should load your diffraction pattern in this interface.

\*Large screenshots may slow down the computation. The recommended screenshot size is between 400 and 800.

**-lower limit:** You need to enter the lower threshold of the double threshold image segmentation in this interface.

**-upper limit:** You need to enter the upper threshold of the double threshold image segmentation in this interface.

\*Canny algorithm adopts the method of double threshold segmentation for region detection. If the pixel value of a region is less than the minimum threshold, it will not be considered. If the pixel value of an area is greater than the maximum threshold, it is retained. If the gradient value of a certain region is between the high and low threshold, the pixel gradient value is found from the 8 neighborhood of the pixel. If there is a pixel gradient value higher than the high threshold, it is retained; if not, it is not considered.

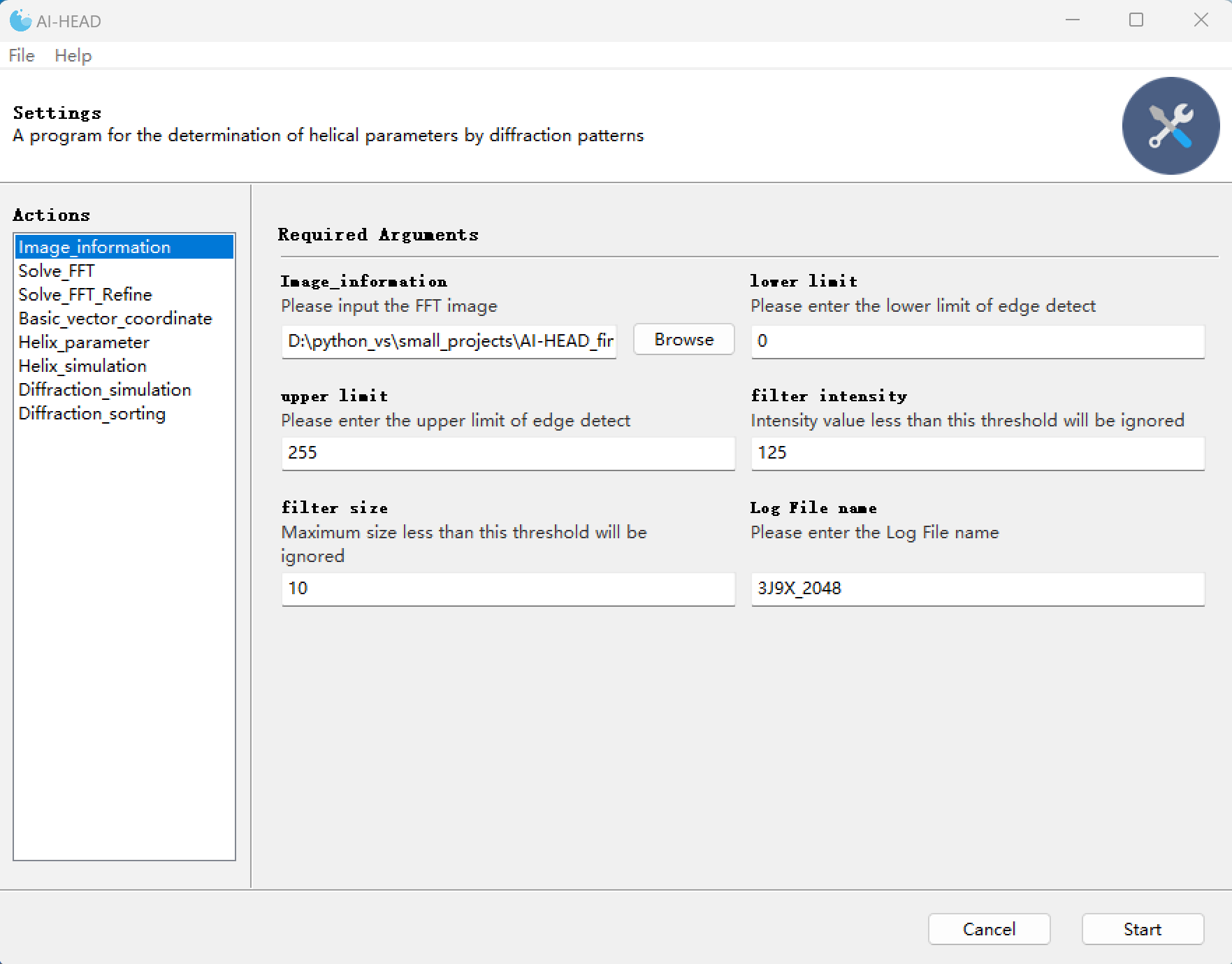
**-filter** **intensity:** You should enter the intensity threshold in this interface.

\*Regions whose intensity below this threshold will be filtered out.

**-filter size:** You should load your diffraction pattern in this interface.

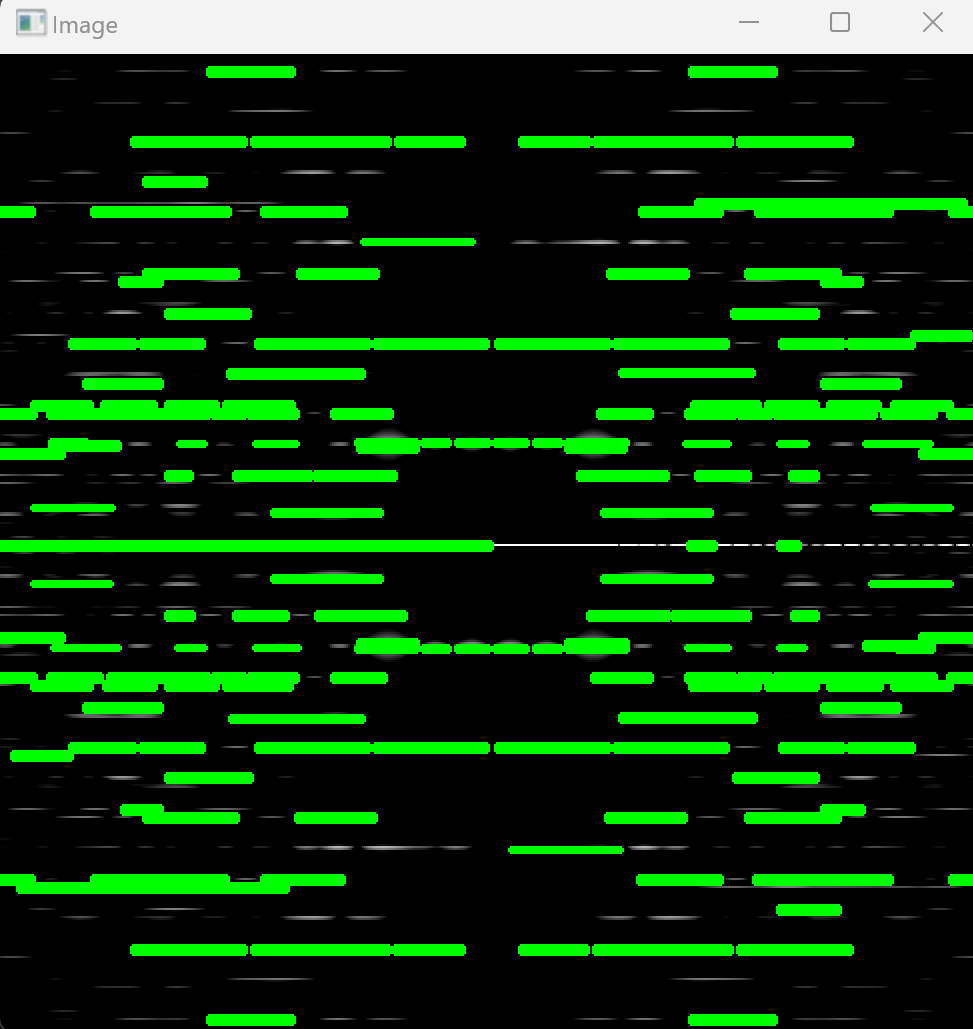
\*Regions whose size below this threshold will be filtered out.

**-Log File name:** This is where you need to enter the name of the log file folder. When you first use this program, it automatically generates a folder named “Image\_information”, and then generate a folder you named under the “Image\_information” folder to record the information. The main information will contain 3 files: the “center” file which contains the center coordinate of the center of diffraction pattern; second, the “region” file which contains all of the range of diffraction region detected; and the edge detection results conducted by Canny algorithm.



Image\_information

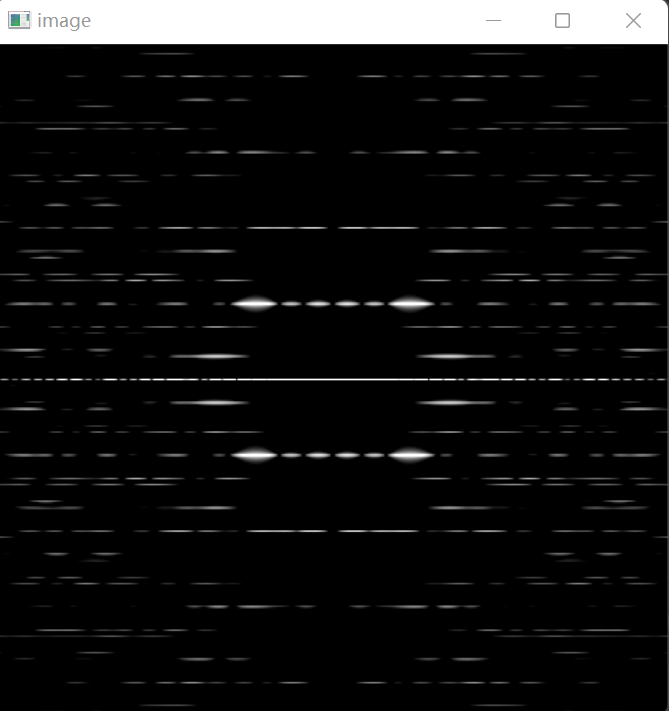
Once you choose the parameters, an interface will pop up to display the results of the edge detection. In this step, Canny algorithm (Canny, 1986) is used to automatically detect the edge of the diffraction pattern and perform the threshold segmentation (N., 1979; A. and S. et al., 2018; D. and H. et al., 2018; Latychevskaia and Fink, 2018; N. and C., 2018; P. and K., 2018). As shown in the figure below, the pink areas are all the actual boundaries detected; The green rectangular area is the actual rectangular area, and the coordinate area of the green area is automatically written to the "region" file. And the result of edge detection will be also saved.



The first interface for displaying the result of edge detection

When you close the second interface, the third interface then pops up to manually supplement areas not detected by the Canny algorithm, or manually segment areas that are not fully segmented

In this case, the diffraction layer lines in the diffraction pattern are very sharp, and the Canny algorithm can find the information of all the diffraction layer lines very accurately, so we could skip this step here. However, when the diffraction pattern is not so clear, the Canny algorithm will not be able to complete the detection and segmentation work well. In that case, manual correction is necessary. You can select undetected areas or areas that are not fully divided by clicking the mouse. These coordinates are then automatically written to the "region" file.



The second interface for manual correction

### Auto-indexing of the diffraction pattern by “global searching”

Now that we've got all of the information of diffraction pattern. Next, AI-HEAD will use this information to conduct auto-indexing with a "global searching" strategy by the “Solve\_FFT” module.

There are 6 inputs in this module:

**-FFT image:** You should load your diffraction pattern in this interface.

\*Large screenshots may slow down the computation. The recommended screenshot size is between 400 and 800.

**-The center coordinates:** You should load the “center” file got in “Image\_information” module.

\*The center file should contain only one row of center coordinate data.

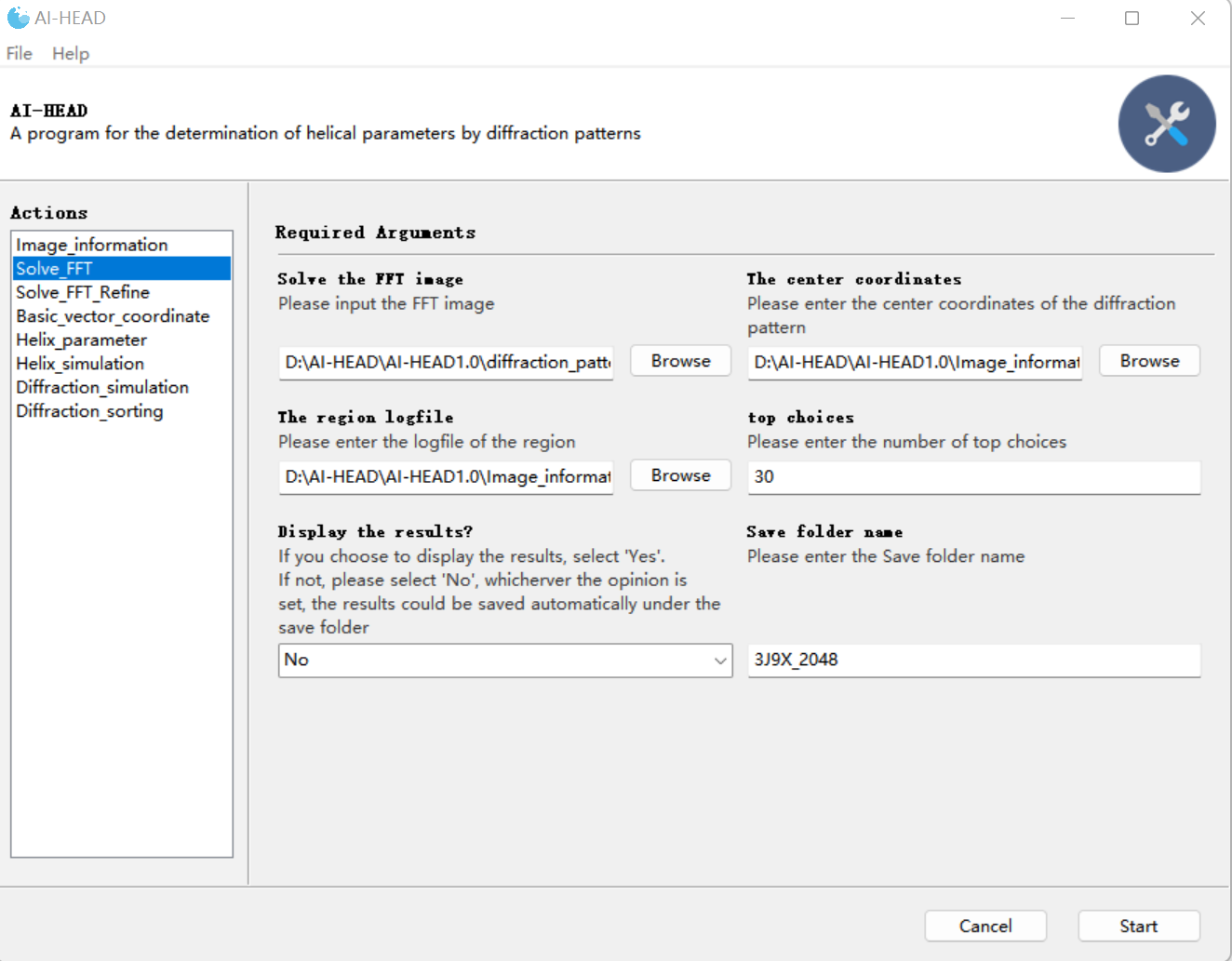
**-The region logfile:** You should load the “region” file got in “Image\_information” module.

**-Top choices:** As AI-HEAD uses a "global searching" strategy, all indexing methods are taken into account. But AI-HEAD will sort all the results according to the conditions of the ideal indexing (such as the number of diffraction layer lines covered under the indexing, the number of diffraction points covered, etc.). Here you can specify the number of optimal AI-HEAD outputs. In most cases, 30 is recommended.

\* In general, indexing at the top of the ranking is ideal. However, as a diffraction pattern can be indexed in many ways, it doesn't mean that results ranking at the bottom is completely wrong. As each pair of basic vectors produces a lattice. When checking the indexing method, the basis vector pairs that produce similar lattices can be considered "equivalent basis vectors". Basic vectors that are farther from the meridian tend to produce multiple sets of possible parameters. So, in summary, when choosing the final indexing method, give priority to the top ones (e.g., top1-5). If several different indexing methods produce similar lattices, then the base vector closer to the meridian and equator is preferred as the final indexing method.

**-Display opinion:** If “Yes” is selected here, the program will bring up a series of screens showing the output of indexing results. No matter what this option selects, the program will save the output in the specified folder.

**-Save folder name:** This is where you need to enter the name of the log file folder. When you first use this program, it automatically generates a folder named “Solve\_FFT”, and then generate a folder you named under the “Solve\_FFT” folder to record the information.



Auto-indexing of the diffraction pattern by “global searching”

After the options are set, the program can be run to conduct “Auto-indexing”. In cases where the diffraction pattern is very clear, “Auto-indexing” is fast and can be completed in less than 1 minute. When the diffraction pattern is very noisy, the speed of indexing is slowed down because the noise is also taken into account in the calculation. In this case, score1 and score2 produce the same lattice, but the position of the basic vectors for score2 is closer to the meridian, which means the number of solution sets calculated by score2 will be smaller, which can reduce the difficulty of subsequent parameter screening. Therefore, we chose score2 as the final indexing method. In this result, different colored dots have different meanings:

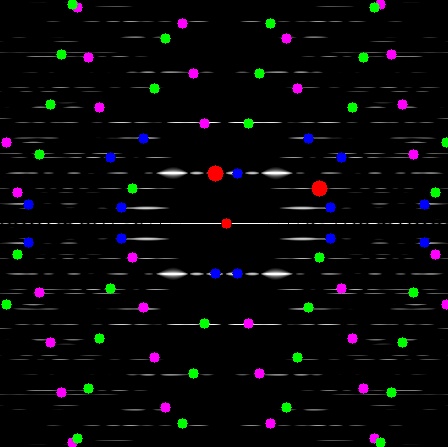
Red point(small): The center of diffraction pattern determined before.

Red points(large): The basic vectors which generate the current lattice.

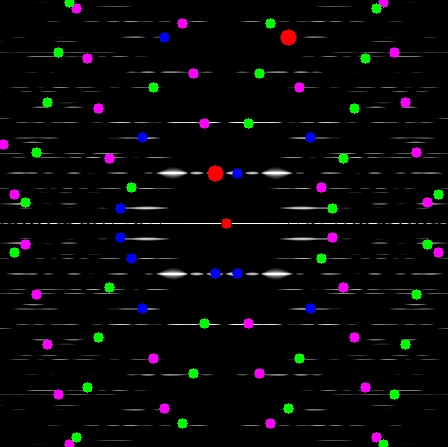
Pink points: The lattice generated by the basic vectors directly.

Green points: The lattice symmetric with the pink lattice about the meridian.

Blue points: Points in lattice or sym-lattice that fall within the diffraction region.



Auto-indexing by “Global seraching” result score 1



Auto-indexing by “Global seraching” result score 2

In this case, since there are lattice points on each diffraction layer line region, the indexing method selected by the "global searching" strategy can be used, and we can skip the "local refinement" step in “Solve\_FFT\_Refine” module. In most cases, just relying on "global searching" strategy can play a good role. However, as "global searching" takes the center of each diffracted patch as the sampling point, if the diffracted patch is very wide, a large deviation will occur and a very ideal solution cannot be obtained. At that point, "Local refinement" can serve as a complement.

### \*Auto-indexing of the diffraction pattern by “local refinement”

\* This step can be skipped when the quality of diffraction pattern is good.

When the quality of our data is not good, "global searching" may not give us the ideal results. In this case, you can use the local refinement policy to adjust the result of “global searching”, which could be achieved by using “Solve\_FFT\_Refine” module.

There are 9 inputs in this module:

**-FFT image:** You should load your diffraction pattern in this interface.

\*Large screenshots may slow down the computation. The recommended screenshot size is between 400 and 800.

**-The center coordinates:** You should load the “center” file got in “Image\_information” module.

\*The center file should contain only one row of center coordinate data.

**-The region logfile:** You should load the “region” file got in “Image\_information” module.

**-Feature point detection algorithm:** In this module, the FAST algorithm is used to find feature points, where you can specify the pattern of finding feature points. See more details in <https://docs.opencv.org/4.x/df/d0c/tutorial_py_fast.html>

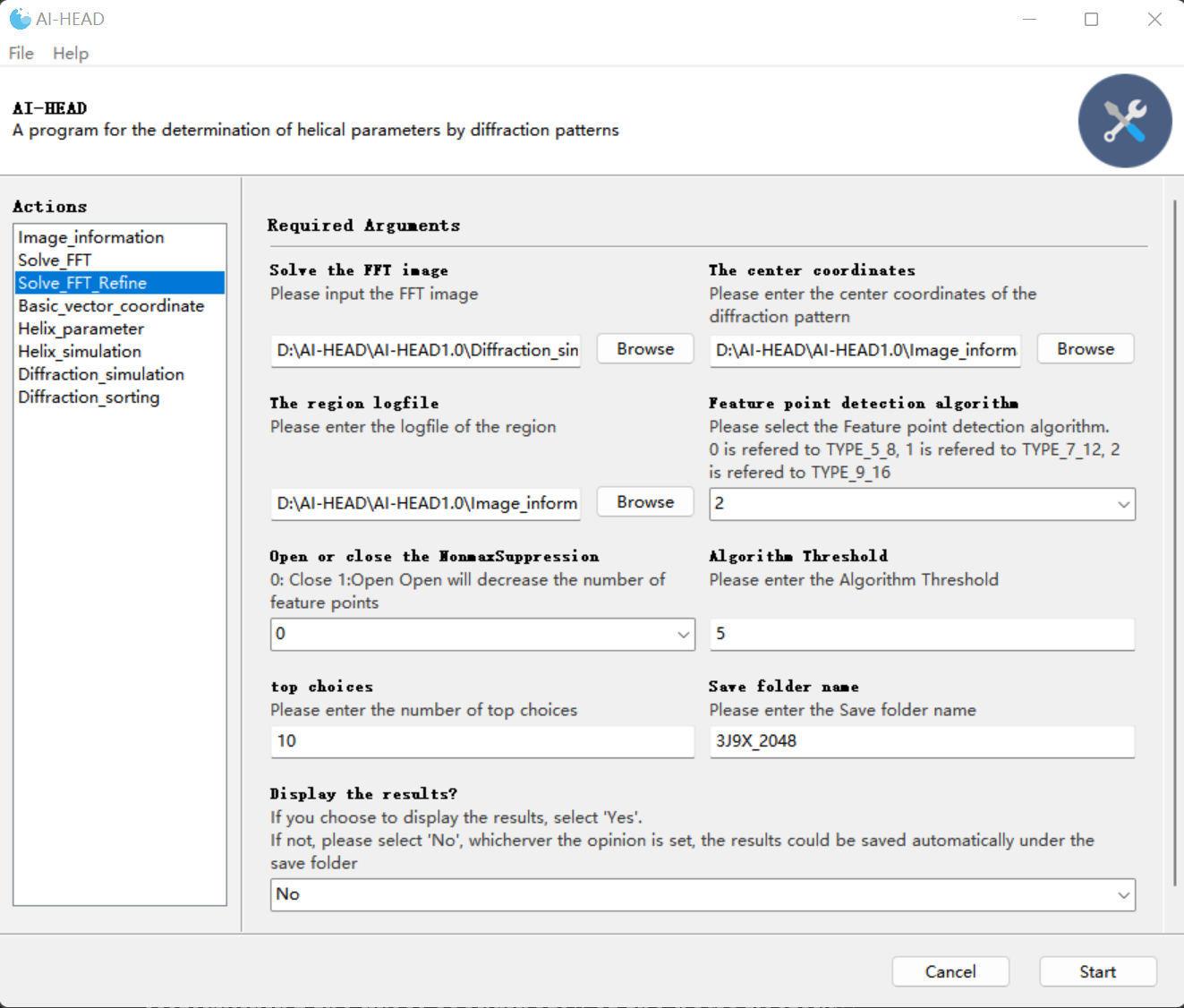
**-Open or close the NonmaxSuppression:** When "Yes" is selected, the program will reduce the number of feature points found, which can reduce the impact of noise to some extent.

**- Algorithm Threshold:** Here you need to specify the threshold at which the algorithm detects the feature points. The smaller the threshold, the more feature points are detected.

**-Top choices:** As AI-HEAD uses a "local refinement" strategy, all indexing methods in the selected region are taken into account. But AI-HEAD will sort all the results according to the conditions of the ideal indexing (such as the number of diffraction points covered.). Here you can specify the number of optimal AI-HEAD outputs. In most cases, 10 is recommended.

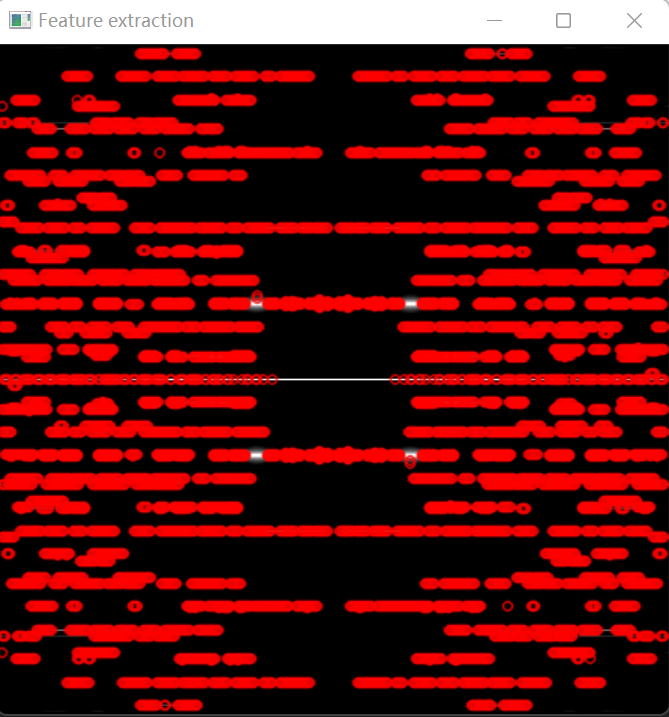
**-Save folder name:** This is where you need to enter the name of the log file folder. When you first use this program, it automatically generates a folder named “Solve\_FFT\_Refinw”, and then generate a folder you named under the “Solve\_FFT\_Refine” folder to record the information.

**-Display opinion:** If “Yes” is selected here, the program will bring up a series of screens showing the output of indexing results. No matter what this option selects, the program will save the output in the specified folder.



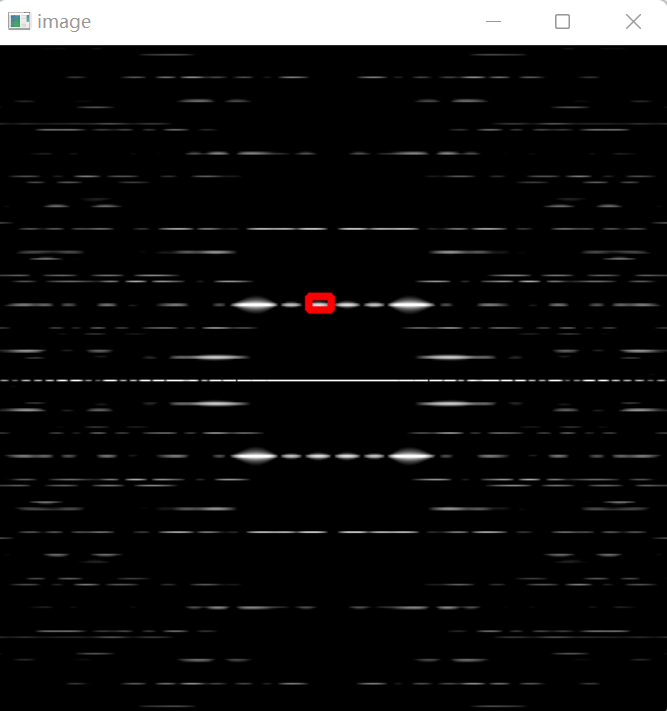
Solve\_FFT\_Refine

The program first provides an interactive interface to display the results of the feature points detection. In this step, FAST algorithm(E. and R. et al., 2010) is used to automatically detect the feature points of the diffraction pattern. All feature points found are highlighted in red.

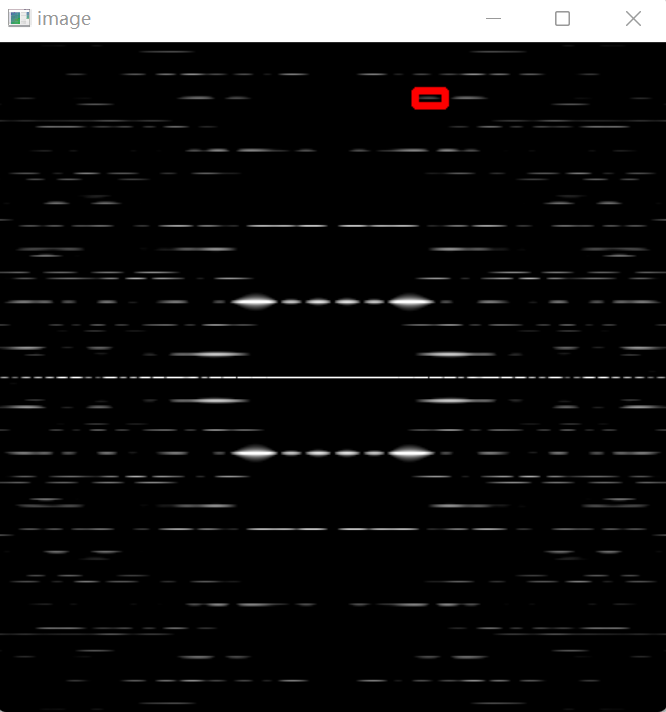


The first interface for displaying the result of FAST

When you are satisfied with the number of feature points, close the first interface, and another interface will pop up to manually delimit the range of the basic vectors. Here, we use the scope selected by the global searching just now as an example:

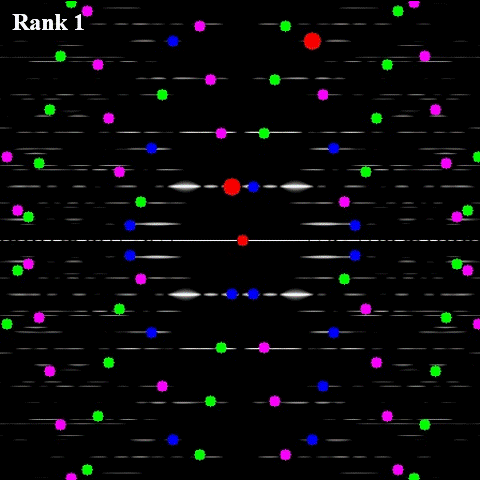


Select the first region manually



Select the second region manually

\* Only two selections can be selected in this step, and it is not recommended to select a large selection, which will increase the calculation time.



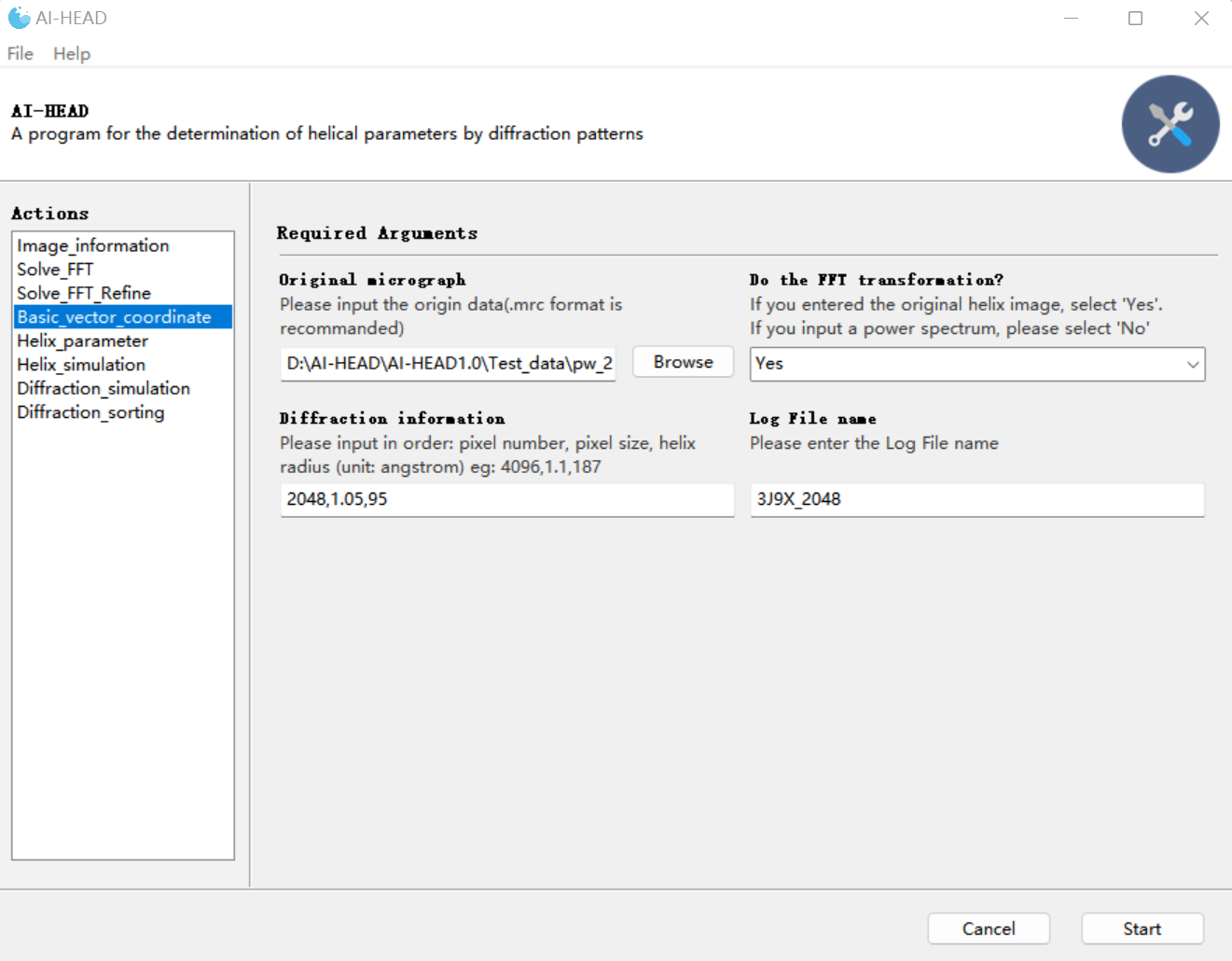
Auto-indexing by “Local refinement” results

As you can see, this step can find the more appropriate position of the basic vector in the two selected ranges and sort it by the number of diffraction patches covered. As “Global searching” strategy only considers the central location of all regions, which could cause some error. At that time, “Local refinement” strategy can serve as a good supplement.

## Calculation of helical parameters

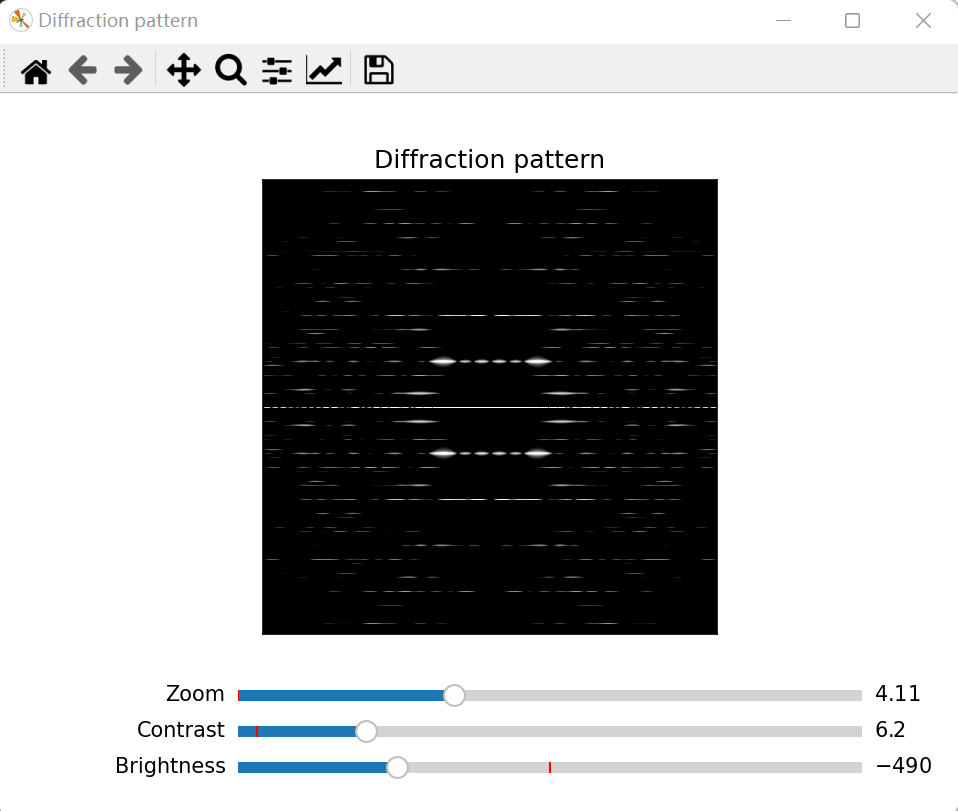
### Determining the “real coordinate” of the basic vectors

Now, AI-HEAD has told you where the basic vectors should be selected for ideal indexing, the indexing process is over, you could calculate the helical parameters based on the indexing results. However, you can't calculate it directly with the coordinates you have. That's because you've zoomed in on the main diffraction area, cut it out and done the whole indexing process. So, we need to go back to the original data and get the real coordinates of the basis vectors. We can still use “Basic\_vector\_coordinate” module to complete this step.



Determining the “real coordinate” of the basic vectors

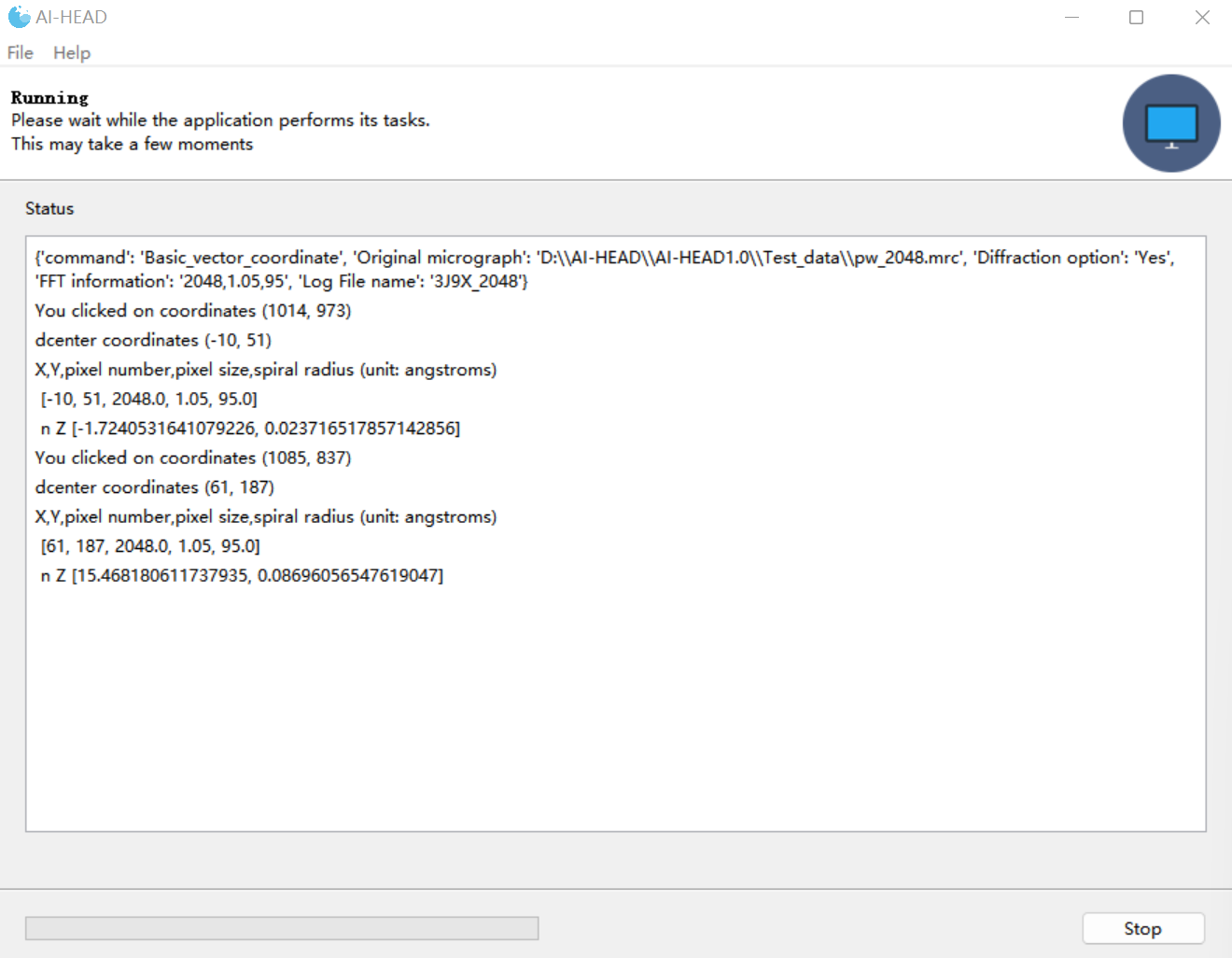
If you have previously obtained a diffraction pattern with AI-HEAD, then all the inputs are consistent with the preparation process. The difference is that we need to double-click in this interface to get the real coordinates of the basic vectors.



Double click here

Double click here

The interface for determining the coordinate of basic vectors



The coordinate will be calculated automatically in terminal

Then, the “real coordinate” of the basic vectors will be automatically recorded in the logfile under the “Basic\_vectors\_coordinate” folder.

### Calculation of helical parameters

Now comes our exciting final step, which is the calculation of the helical parameters. This step could be finished by the “Helix\_parameter” module.

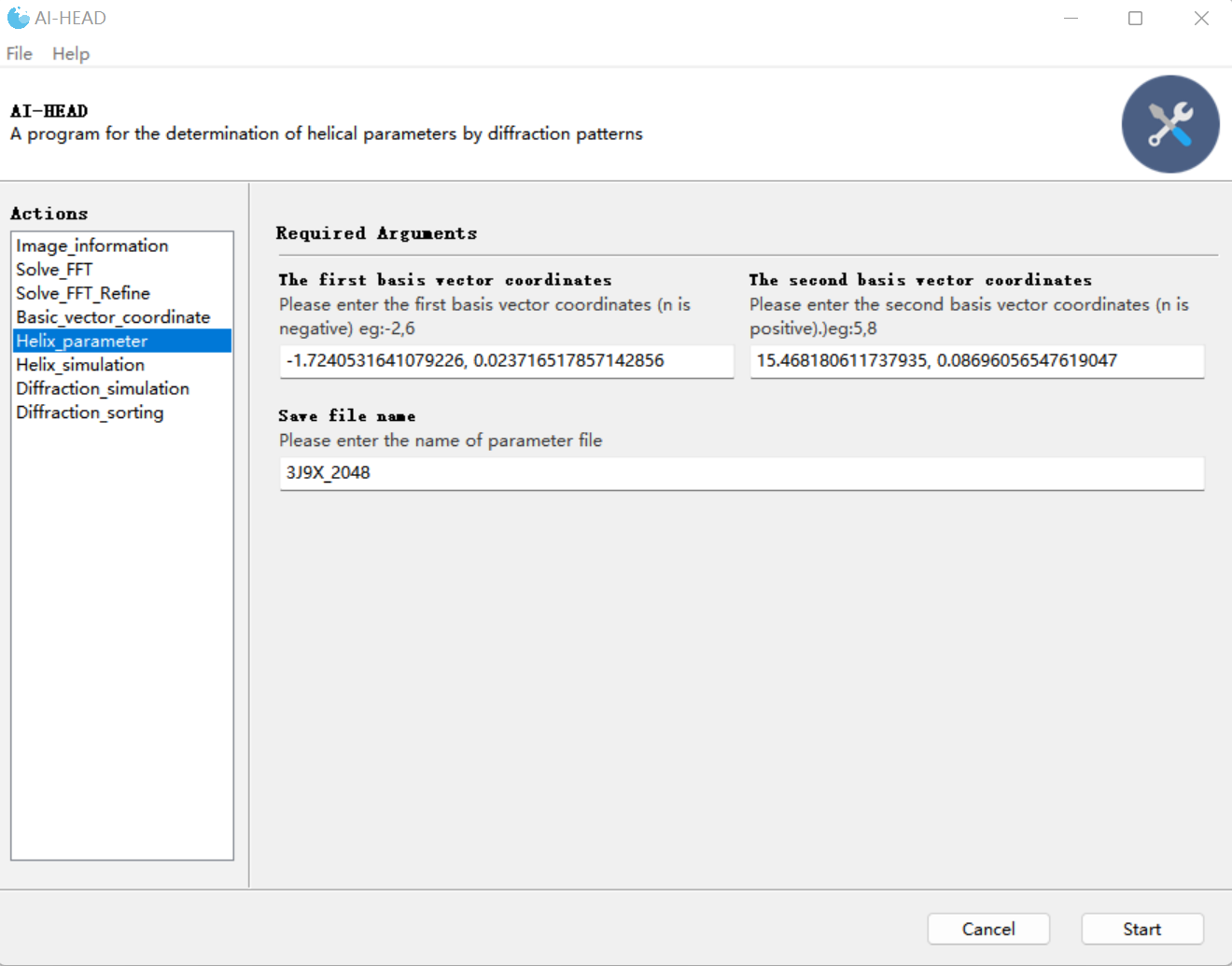
There are 3 inputs in this module:

**-The first basis vector coordinate:** You should input the coordinate got before in “Basic\_vector\_coordinate” module whose n is negative.

**-The second basis vector coordinate:** You should input the coordinate got before in “Basic\_vector\_coordinate” module whose n is positive.

\*The first input box n must be negative, and the second must be positive, otherwise there will be no output.

**-Save file name:** This is where you need to enter the name of the log file. When you first use this program, it automatically generates a folder named “Helix\_parameter”, and then generate two charts you named under the “Helix\_parameter” folder to record the information.



The calculation of helical parameters

Finally, you get a set of solution sets that contain all the possibilities of helical parameters. In this case, we can see that the parameter highlighted in red is very close to the correct parameters. recorded in EMDB (2.9Å, 24.5°).

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| [n1,n2] | rise | P | twist | N | k\_value | [Z1,Z2] | [x1\_coff] | [x2\_coff] |
| [-1, 1] | 9.04 | 42.16 | 77.14 | 1 | [1, 0] | ['0.02372', '0.08696'] | [-1.0, 1.0, 0] | [1.0, -1.0, 1] |
| [-1, 2] | 7.44 | 42.16 | 63.53 | 1 | [1, 0] | ['0.02372', '0.08696'] | [-1.0, 1.0, 0] | [2.0, -2.0, 1] |
| [-1, 3] | 6.32 | 42.16 | 54 | 1 | [1, 0] | ['0.02372', '0.08696'] | [-1.0, 1.0, 0] | [3.0, -3.0, 1] |
| [-1, 4] | 5.5 | 42.16 | 46.96 | 1 | [1, 0] | ['0.02372', '0.08696'] | [-1.0, 1.0, 0] | [4.0, -4.0, 1] |
| [-1, 5] | 4.87 | 42.16 | 41.54 | 1 | [1, 0] | ['0.02372', '0.08696'] | [-1.0, 1.0, 0] | [5.0, -5.0, 1] |
| [-1, 6] | 4.36 | 42.16 | 37.24 | 1 | [1, 0] | ['0.02372', '0.08696'] | [-1.0, 1.0, 0] | [6.0, -6.0, 1] |
| [-1, 7] | 3.95 | 42.16 | 33.75 | 1 | [1, 0] | ['0.02372', '0.08696'] | [-1.0, 1.0, 0] | [7.0, -7.0, 1] |
| [-1, 8] | 3.61 | 42.16 | 30.86 | 1 | [1, 0] | ['0.02372', '0.08696'] | [-1.0, 1.0, 0] | [8.0, -8.0, 1] |
| [-1, 9] | 3.33 | 42.16 | 28.42 | 1 | [1, 0] | ['0.02372', '0.08696'] | [-1.0, 1.0, 0] | [9.0, -9.0, 1] |
| [-1, 10] | 3.09 | 42.16 | 26.34 | 1 | [1, 0] | ['0.02372', '0.08696'] | [-1.0, 1.0, 0] | [10.0, -10.0, 1] |
| [-1, 11] | 2.87 | 42.16 | 24.55 | 1 | [1, 0] | ['0.02372', '0.08696'] | [-1.0, 1.0, 0] | [11.0, -11.0, 1] |
| [-1, 12] | 2.69 | 42.16 | 22.98 | 1 | [1, 0] | ['0.02372', '0.08696'] | [-1.0, 1.0, 0] | [12.0, -12.0, 1] |
| [-1, 13] | 2.53 | 42.16 | 21.6 | 1 | [1, 0] | ['0.02372', '0.08696'] | [-1.0, 1.0, 0] | [13.0, -13.0, 1] |
| [-1, 14] | 2.39 | 42.16 | 20.38 | 1 | [1, 0] | ['0.02372', '0.08696'] | [-1.0, 1.0, 0] | [14.0, -14.0, 1] |
| [-1, 15] | 2.26 | 42.16 | 19.29 | 1 | [1, 0] | ['0.02372', '0.08696'] | [-1.0, 1.0, 0] | [15.0, -15.0, 1] |
| [-1, 16] | 2.14 | 42.16 | 18.31 | 1 | [1, 0] | ['0.02372', '0.08696'] | [-1.0, 1.0, 0] | [16.0, -16.0, 1] |
| [-2, 1] | 5.06 | 11.5 | 158.4 | 1 | [0, 1] | ['0.02372', '0.08696'] | [-2.0, -2.0, 1] | [1.0, 1.0, 0] |
| [-2, 2] | 9.04 | 84.33 | 38.57 | 2 | [1, 0] | ['0.02372', '0.08696'] | [-1.0, 1.0, 0] | [1.0, -1.0, 1] |
| [-2, 3] | 4.08 | 9.04 | 162.58 | 1 | [1, 1] | ['0.02372', '0.08696'] | [-2.0, -2.0, 1] | [3.0, 3.0, -1] |
| [-2, 4] | 7.44 | 84.33 | 31.76 | 2 | [1, 0] | ['0.02372', '0.08696'] | [-1.0, 1.0, 0] | [2.0, -2.0, 1] |
| [-2, 5] | 3.42 | 7.44 | 165.41 | 1 | [2, 1] | ['0.02372', '0.08696'] | [-2.0, -2.0, 1] | [5.0, 5.0, -2] |
| [-2, 6] | 6.32 | 84.33 | 27 | 2 | [1, 0] | ['0.02372', '0.08696'] | [-1.0, 1.0, 0] | [3.0, -3.0, 1] |
| [-2, 7] | 2.94 | 6.32 | 167.44 | 1 | [3, 1] | ['0.02372', '0.08696'] | [-2.0, -2.0, 1] | [7.0, 7.0, -3] |
| [-2, 8] | 5.5 | 84.33 | 23.48 | 2 | [1, 0] | ['0.02372', '0.08696'] | [-1.0, 1.0, 0] | [4.0, -4.0, 1] |
| [-2, 9] | 2.58 | 5.5 | 168.98 | 1 | [4, 1] | ['0.02372', '0.08696'] | [-2.0, -2.0, 1] | [9.0, 9.0, -4] |
| [-2, 10] | 4.87 | 84.33 | 20.77 | 2 | [1, 0] | ['0.02372', '0.08696'] | [-1.0, 1.0, 0] | [5.0, -5.0, 1] |
| [-2, 11] | 2.3 | 4.87 | 170.18 | 1 | [5, 1] | ['0.02372', '0.08696'] | [-2.0, -2.0, 1] | [11.0, 11.0, -5] |
| [-2, 12] | 4.36 | 84.33 | 18.62 | 2 | [1, 0] | ['0.02372', '0.08696'] | [-1.0, 1.0, 0] | [6.0, -6.0, 1] |
| [-2, 13] | 2.07 | 4.36 | 171.15 | 1 | [6, 1] | ['0.02372', '0.08696'] | [-2.0, -2.0, 1] | [13.0, 13.0, -6] |
| [-2, 14] | 3.95 | 84.33 | 16.88 | 2 | [1, 0] | ['0.02372', '0.08696'] | [-1.0, 1.0, 0] | [7.0, -7.0, 1] |
| [-2, 15] | 1.89 | 3.95 | 171.94 | 1 | [7, 1] | ['0.02372', '0.08696'] | [-2.0, -2.0, 1] | [15.0, 15.0, -7] |
| [-2, 16] | 3.61 | 84.33 | 15.43 | 2 | [1, 0] | ['0.02372', '0.08696'] | [-1.0, 1.0, 0] | [8.0, -8.0, 1] |

The main chart of results

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| [n1,n2] | rise | P | twist | N | k\_value | [Z1,Z2] | [x1\_coff] | [x2\_coff] |
| [-1, 1] | 9.04 | 11.5 | 282.86 | 1 | [0, 1] | ['0.02372', '0.08696'] | [-1.0, -1.0, 1] | [1.0, 1.0, 0] |
| [-2, 2] | 9.04 | 23 | 141.43 | 2 | [0, 1] | ['0.02372', '0.08696'] | [-1.0, -1.0, 1] | [1.0, 1.0, 0] |

The additional chart of results

## Auto-simulation of diffraction pattern

### Generate a series of simulated projections based on the helical parameters

However, you do not actually know which one is the correct solution in the solution set. The correct parameters can only be obtained by using IHRSR for helical reconstruction based on all the parameters in turn. When the solution set is very large, this can be a very time-consuming task. Therefore, AI-HEAD provides a function to achieve a simple screening of helical parameters by comparing the diffraction pattern and lattice based on all the parameters calculated before with that of the original data. So first of all, you need to get the projections based on all of the helical parameters, which could be easily done by the “Helix\_simulation” module.

There are 6 inputs in this module:

**-Helix parameter:** You should load the chart of helical parameters in this interface.

**-Helix radius:** Since we need to simulate a series of helical projections similar to the original data, here you need to give the radius of the original data (in angstrom).

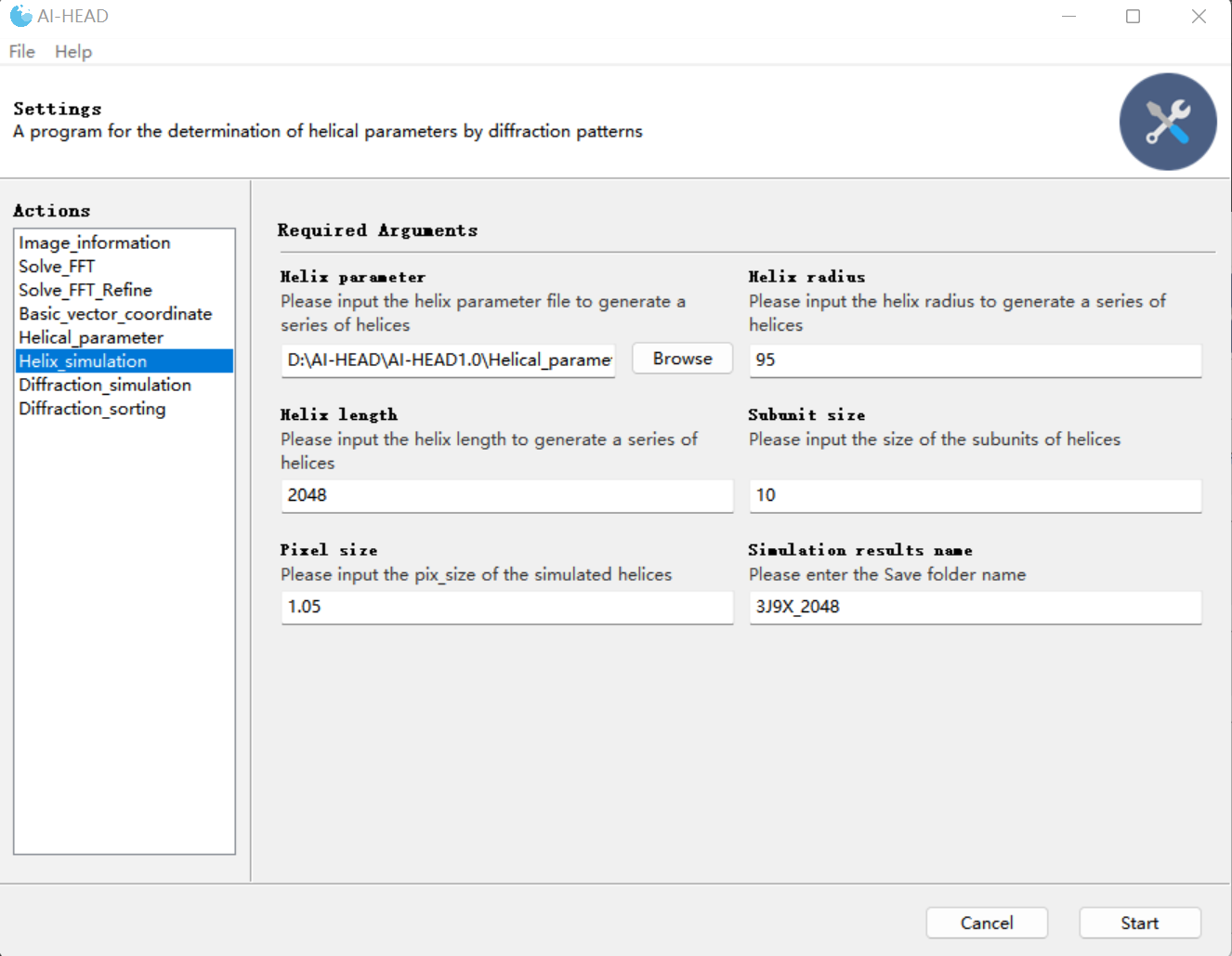
-**Helix length:** Here you need to input the length of the simulated helices, preferably the same as the length of the original data.

**-Subunit size:** Here you need to input the size of each unit of the simulated helices.

**-Pixel size:** As mentioned earlier, you need to input the pixel size of the original data here to guarantee that the pixel size of the simulated data is the same as the original data.

**-Simulation results name:** This is where you need to enter the name of the log file. When you first use this program, it automatically generates a folder named “Helix\_simulation”, and then generate a folder you named under the “Helix\_simulation” folder, a folder named "projection" can then be generated to save a series of projections and a “parameter” file.

\*In this step, the program generates a series of projections based on the parameters. At the same time, the program will also generate the layer line information of the diffraction space according to the parameter information and write it into the "Parameter" file, which will be used for the subsequent generation of the simulation lattice.



Helix\_simulation

### Get the diffraction pattern of simulated projections

Now, you’ve got the projections based on all of the parameters calculated before. Next, you need to get the corresponding diffraction pattern from these projections, which could be achieved by using “Diffraction\_simulation” module.

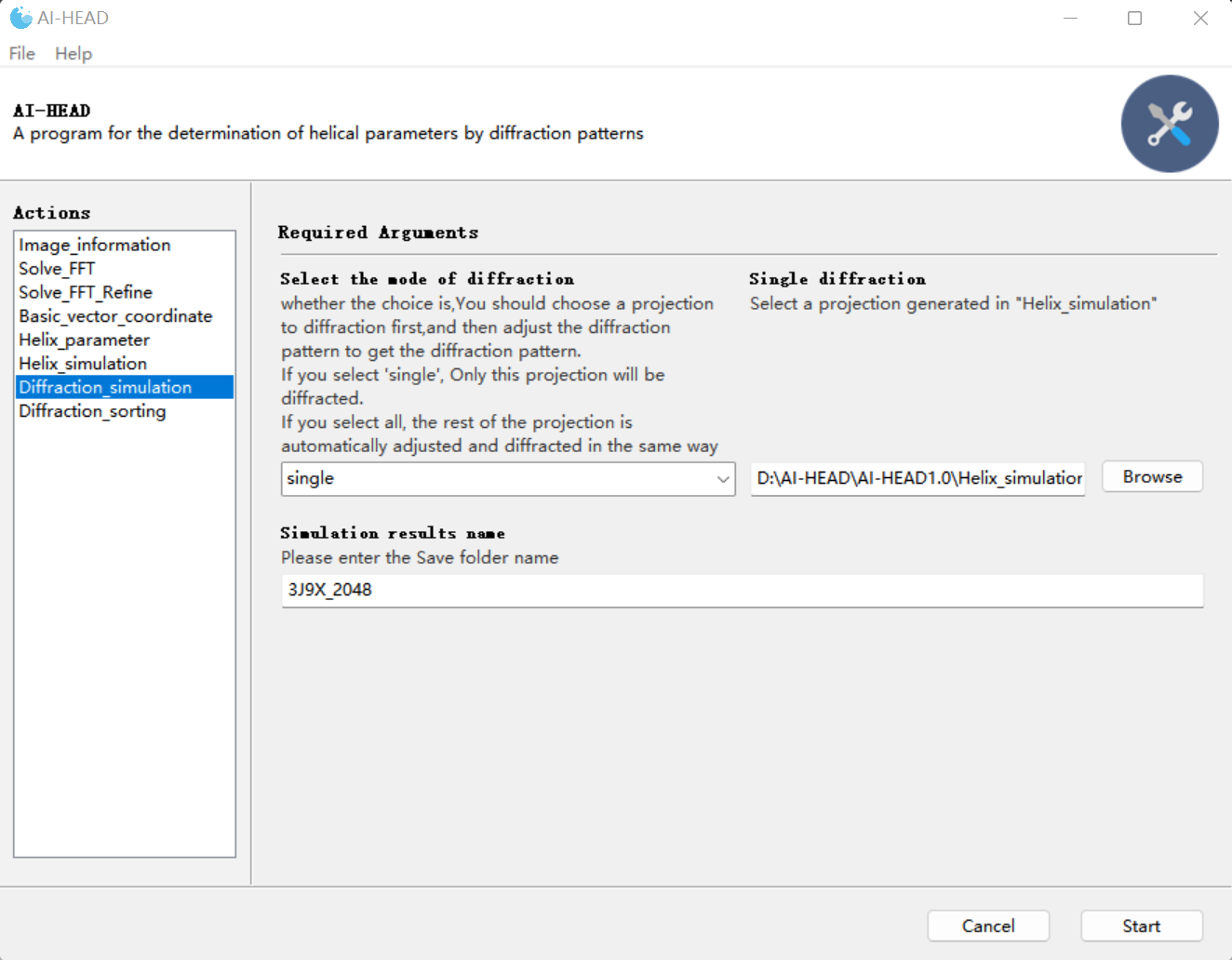
There are 3 inputs in this module:

**-Diffraction mode:**There are "single" and "All" modes in this module. Since it is required to adjust the diffraction power spectrum before obtaining the diffraction pattern, the user can first adjust one of the projections in the "single" mode to obtain a more suitable diffraction pattern. Meanwhile, information such as the adjusted brightness, contrast, and the region of diffraction pattern can be exported to a "value" file. Next, you can switch to the "All" mode, and the program will automatically diffract all the remaining projections according to the information adjusted in the "single" mode and output them in batch. You could modify the data in the "value" file to achieve different adjustment methods until a more ideal diffraction pattern is obtained.

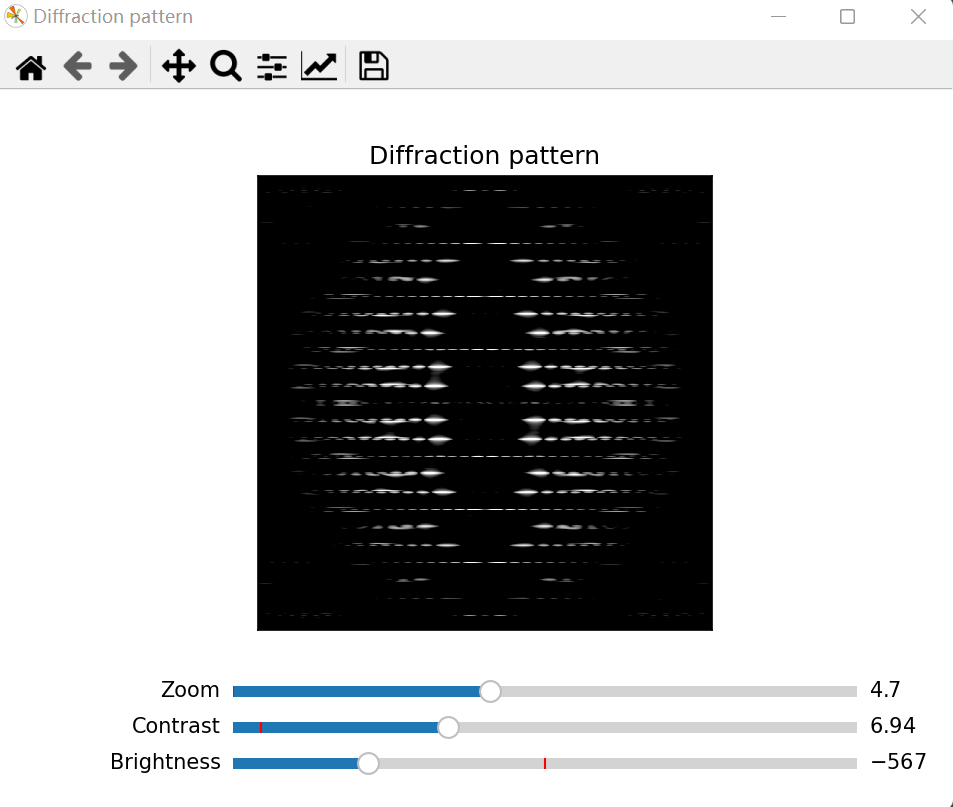
**-Single diffraction:** In “single” mode, you need to load one of the projections generated in “Helix\_simulation”.

**-Simulation results name:** This is where you need to enter the name of the log file. When you first use this program, it automatically generates a folder named “Diffraction\_simulation”, and then generate a folder you named under the “Diffraction\_simulation” folder to record the information. \***Note that the name here must be the same as the previous name in “Helix\_simulation”!**

First, let’s select the "single" mode and select one of the simulated projections to conduct the adjustment.

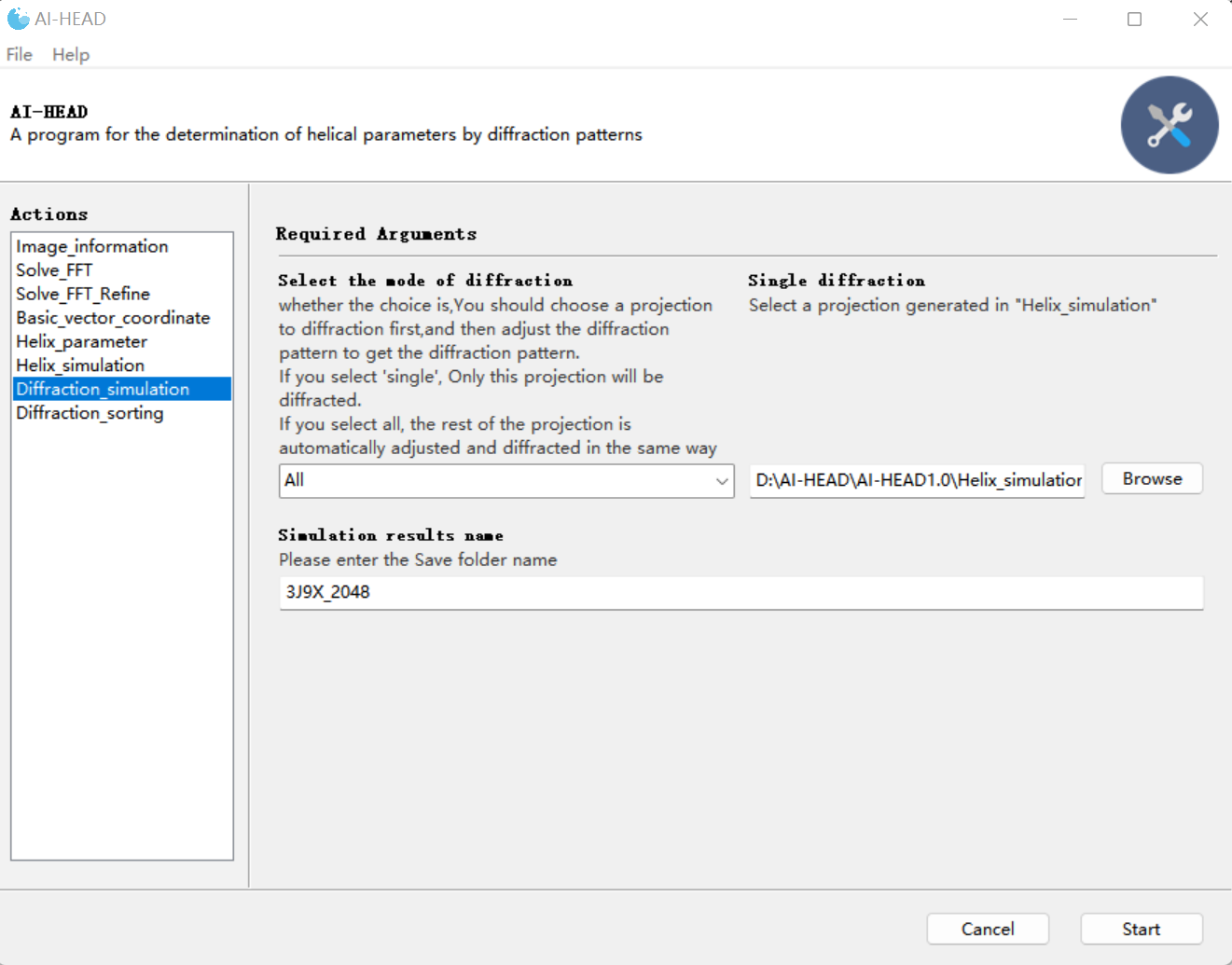


“single” mode in Diffraction\_simulation

After you have finished the adjustment of the diffraction pattern, press the space bar again to obtain the diffraction pattern of the single projection and a file named “value” which contains all the information about the adjustment.

Adjust the power spectrum to get the diffraction pattern

However, what we really need is "Contrast" and "Brightness" information, because in order to compare with the original data, the size of the diffraction region needs to be the same as the original data. If you have previously obtained a diffraction pattern using AI-HEAD, then you have also obtained a “value” file that describes the adjustment of the original data. Now you only need to change the data corresponding to “region” in the current “value” file to be consistent with “region” in the “value” file of the original data, and then switch to the “All” mode. The diffraction pattern is then automatically diffraction pattern for all the projections. Then, all diffraction patterns of simulated projections will be adjusted in the same way. You can manually adjust the values of "Contrast" and "Brightness" in the "value" file and reprogram until all diffraction patterns are clear.



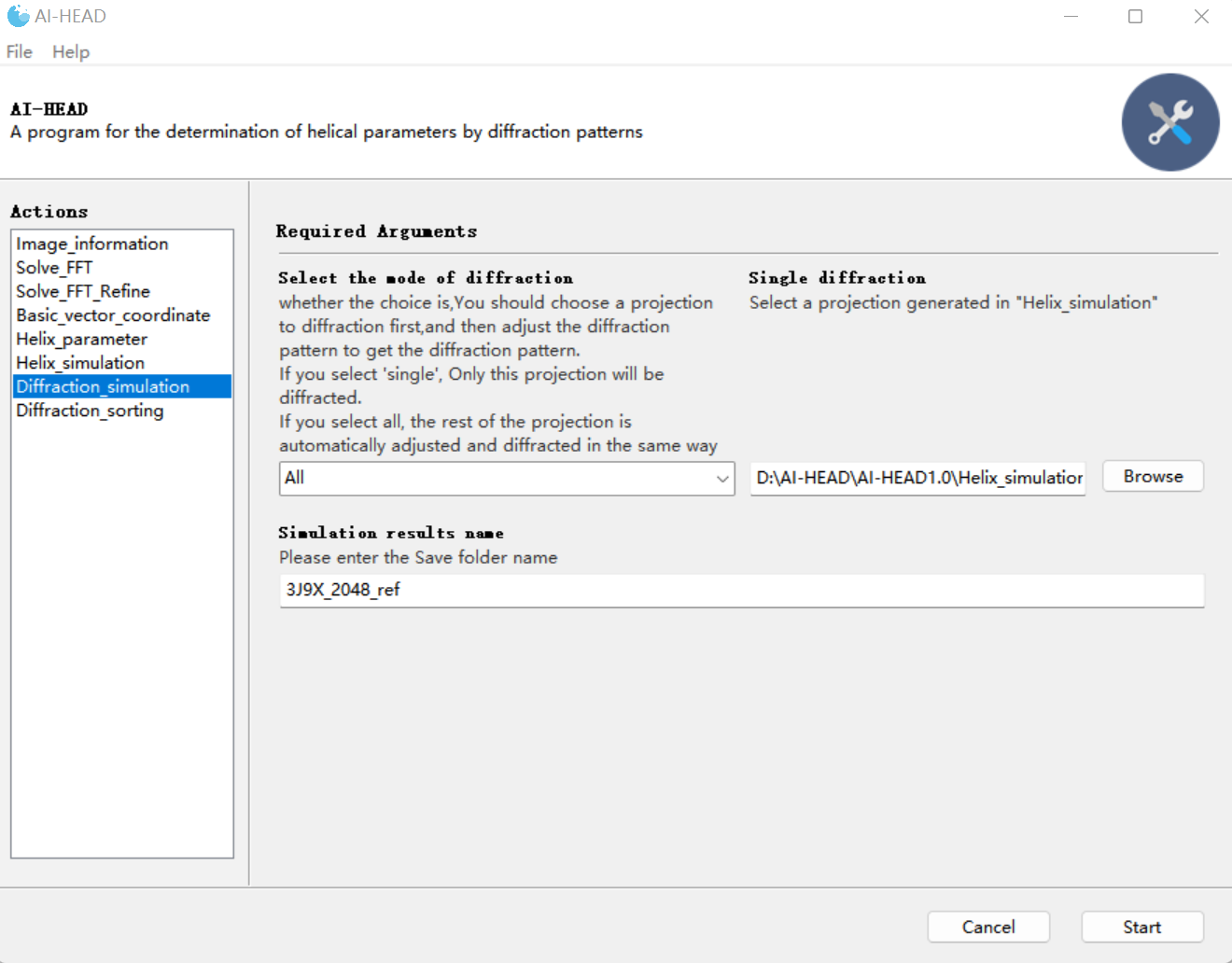
“All” mode in Diffraction\_simulation

If your diffraction pattern is not obtained from the AI-HEAD, you can reuse the AI-HEAD in this step to generate a diffraction pattern from the original data as a reference. In this case, you only need to keep the reference the same scale as the simulated diffraction pattern.

The steps are as follows:

1. Create a directory under “Helix\_simulation”, for example, called "3J9X\_ref".
2. Create another directory "projection" in the directory you created.
3. Put your original data under "projection".
4. Create a directory under “Diffraction\_simulation” called "3J9X\_ref" (It needs to be the same as the name created earlier in "Helix\_simulation"), and copy the “value” file of the simulated data to the directory
5. Select the "All" mode and select the original data stored in "Helix\_simulation" in the single diffraction, the file name remains the same as before.

You can manually adjust the values of "Contrast" and "Brightness" in the "value" file and reprogram until the diffraction pattern is clear.



“All” mode to get the diffraction pattern of original data

## Sorting the results

Now we have simulated the diffraction pattern corresponding to all parameters and obtained their layer line information. Then we need to compare the simulated data with the original data. This function can be completed by "Diffraction\_sorting" module.

There are 6 inputs in this module:

**-Reference image:** You need to load the diffraction pattern of original data here.

**-Simulation\_image\_path:** You need to load the path where the diffraction patterns of simulation data were saved.

**-Diffraction features:** You need to load the “parameter” file got in “Helix\_simulation” here.

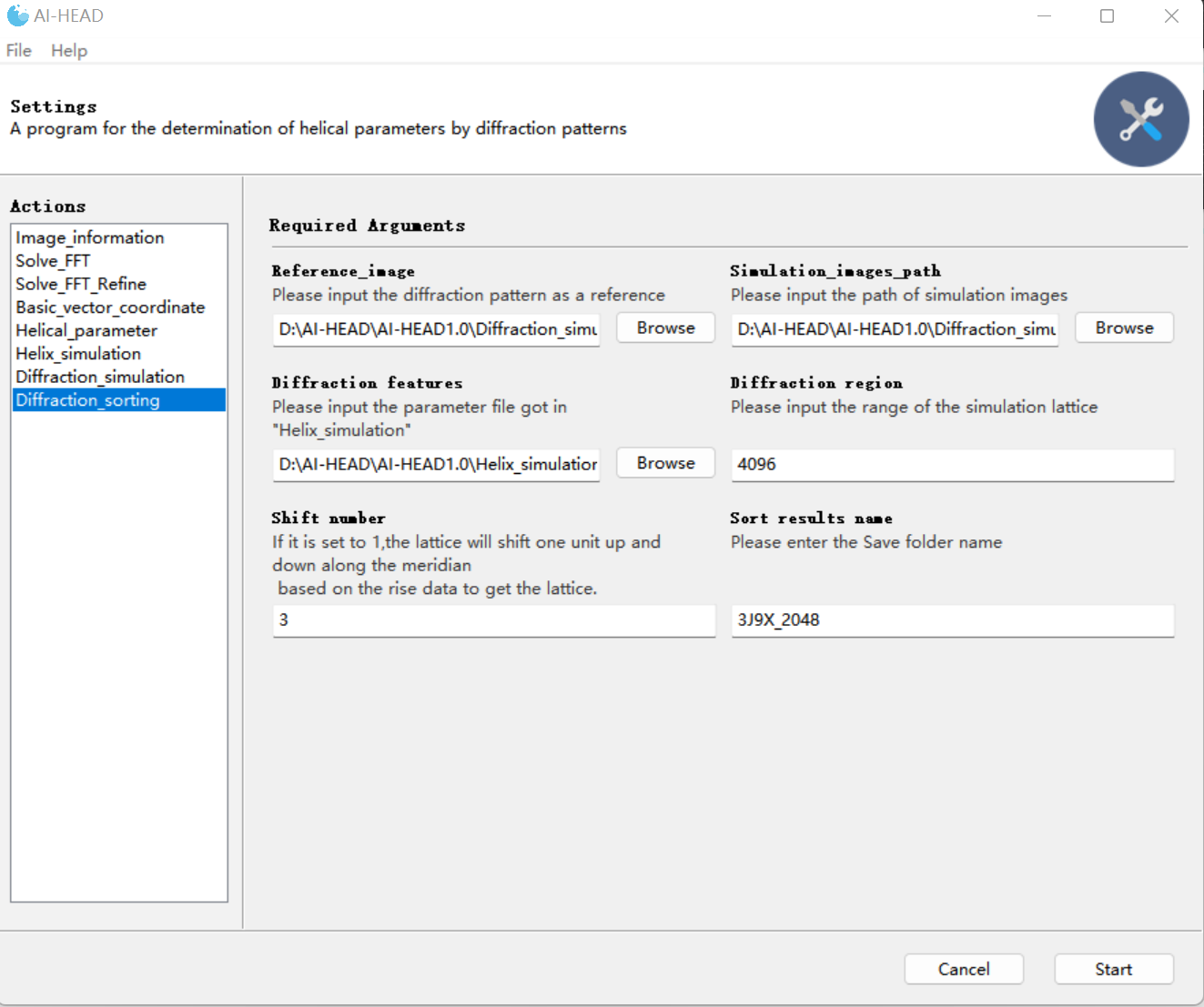
**-Diffraction region:** Here you need to specify the size of the range to generate the simulated lattice.

\* The larger the range, the more lattice points are generated and the longer the calculation time. When the number of points in the result is small, you can consider increasing this value.

**-Shift number:** Since the actual diffraction pattern is obtained by a single simulated lattice translating several times along the meridian, the number of simulated lattice translations is specified here.

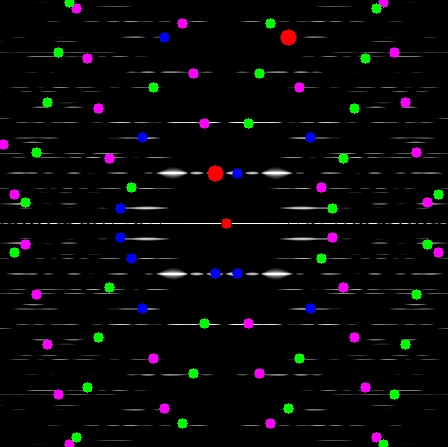
\* The maximum value can only be set to 4, 3 is recommended here.

**-Sort results name:** This is where you need to enter the name of the log file. When you first use this program, it automatically generates a folder named “Diffraction\_sorting”, and then generate a folder you named under the “Diffraction\_simulation” folder, the "Converge" and "Compare" folders can then be generated. The "Converge" folder stores a series of fusion graphs of simulated data and original data; In the "Compare" folder, the simulation lattice is drawn using the layer line information based on the fusion graph.

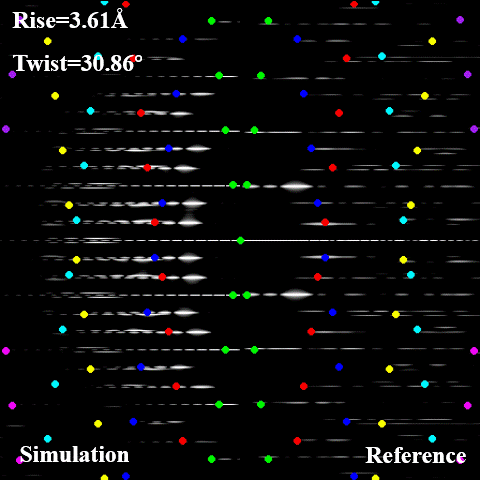


Diffraction\_sorting

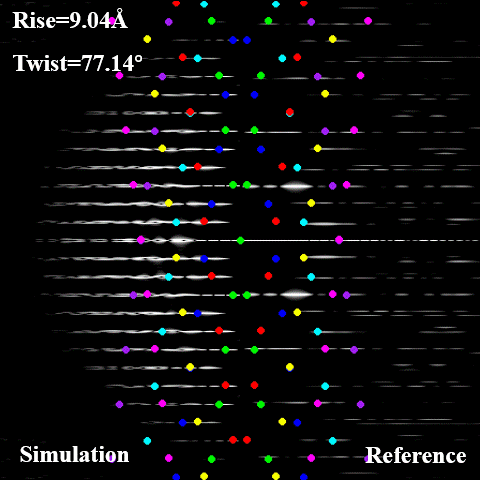
Finally, based on results with "Auto-indexing", we can divide simulation data into three categories: similar to the “Auto-indexing” result is called "good", partially similar is called "uncertain," and completely different is called "bad". To be conservative, we can simply exclude the "bad" data completely. In this case, 18 of the 34 sets of results were removed. Then the remaining data can be used to conduct the helical reconstruction using IHRSR to get the unique correct solution.



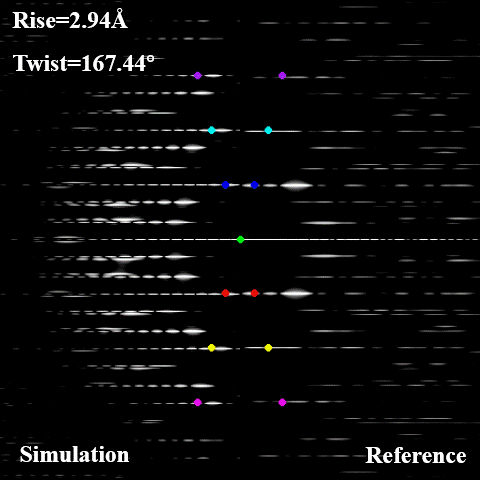
Auto-indexing result



“Good” results



“Bad” results



“Uncertain” results

# The tutorial of auto-indexing of helical data(2D average power spectrum) in EMDB

## Auto-indexing of diffraction pattern

### Preparing the diffraction pattern of the helix

Now, let’s begin our work. Here's a simple tutorial using the sample data (MAVS CARD EMD-6428). First of all, you need to get the diffraction pattern of the helix. You could finish this step by using “Basic\_vector\_coordinate” module in AI-HEAD.

There are 4 inputs in this module:

**-Original micrograph:** You could load your data (in mrc format) in this interface.

\*The data provided here requires equal length and width, 1024 or 2048 is recommended. The longer the helix, the sharper the diffraction layer lines, and the more accurate the indexing results.

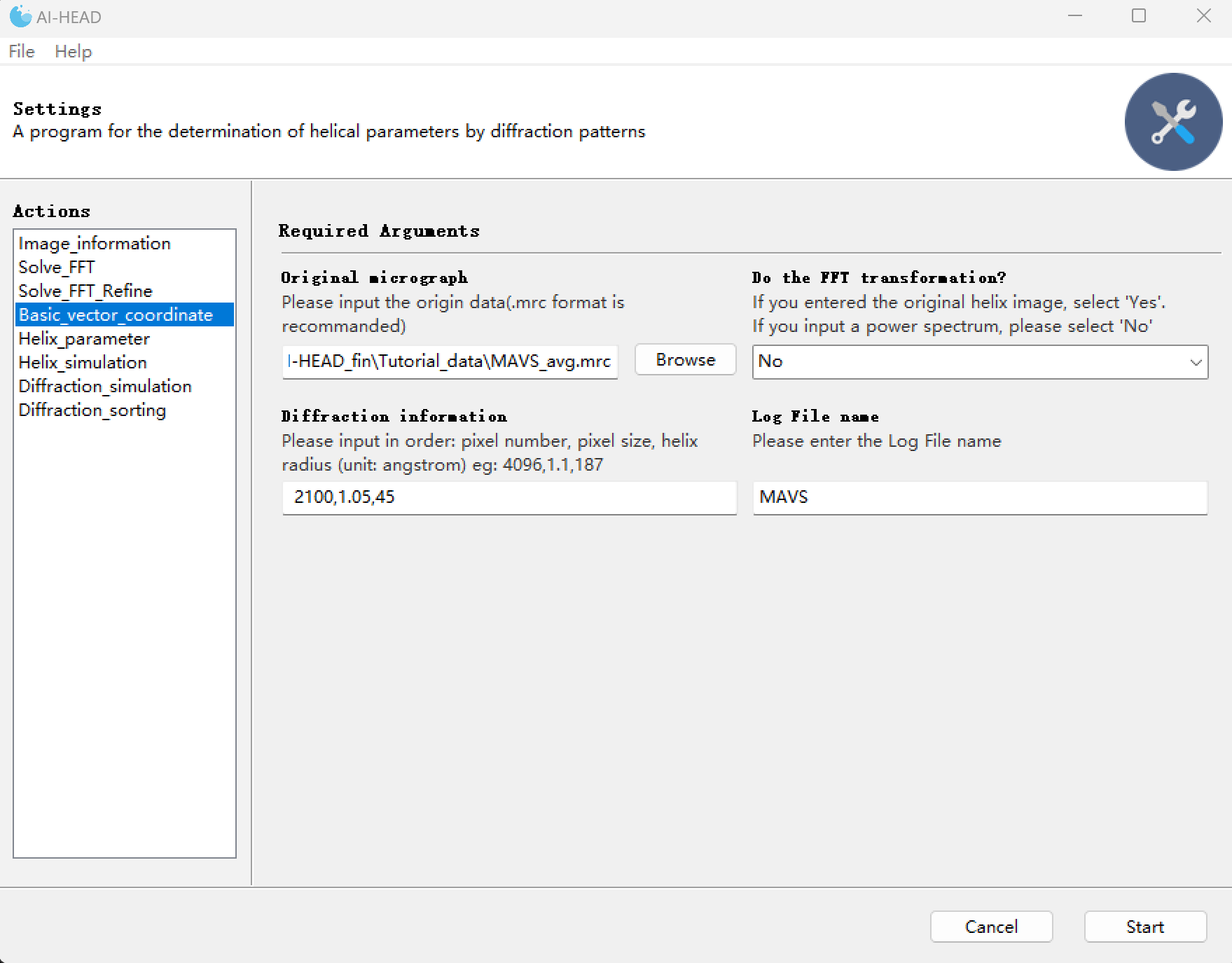
**-FFT opinion:** Since the process of indexing is conducted in the Fourier space. If you provide a projection of the helix, you need to choose “yes” here, and AI-HEAD will do the Fourier transformation automatically; if you provide the power spectrum of the projection of helix, then you just choose “No” here.

**-Diffraction information:** Here you need to tell this program the basic information of your data, which is consist of the size of the data, the pixel size of the data and the radius of the helix (in angstrom).

\*These data are separated by commas.

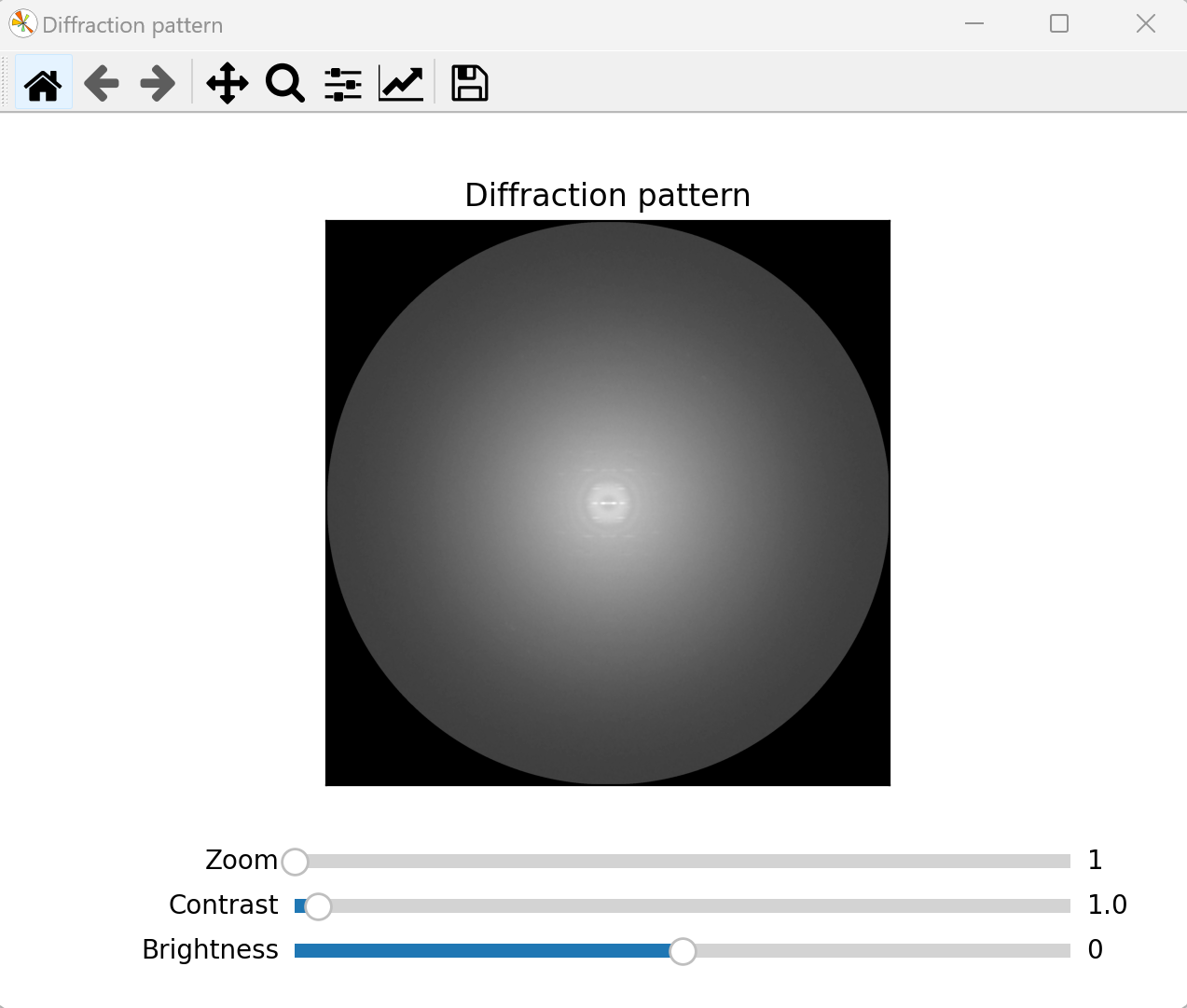
**-Log file name:** This is where you need to enter the name of the log file. Since you need to find the coordinates of the base vector for indexing results here, this step can be done with one click of the interface. The corresponding coordinate information is automatically recorded in the log file.

\*In fact, we just check the diffraction pattern here, so no coordinate information will be generated.



Get the diffraction pattern of the helix

Once we have entered all the information, we can click "Start" to run, and the program will generate an interface of the diffraction pattern:



The 2D average power spectrum

Since the power spectrum obtained by direct Fourier transform can be noisy, and indexing requires only a few major layer lines based on the low frequency region for analysis, the program provides three sliders to adjust the image.

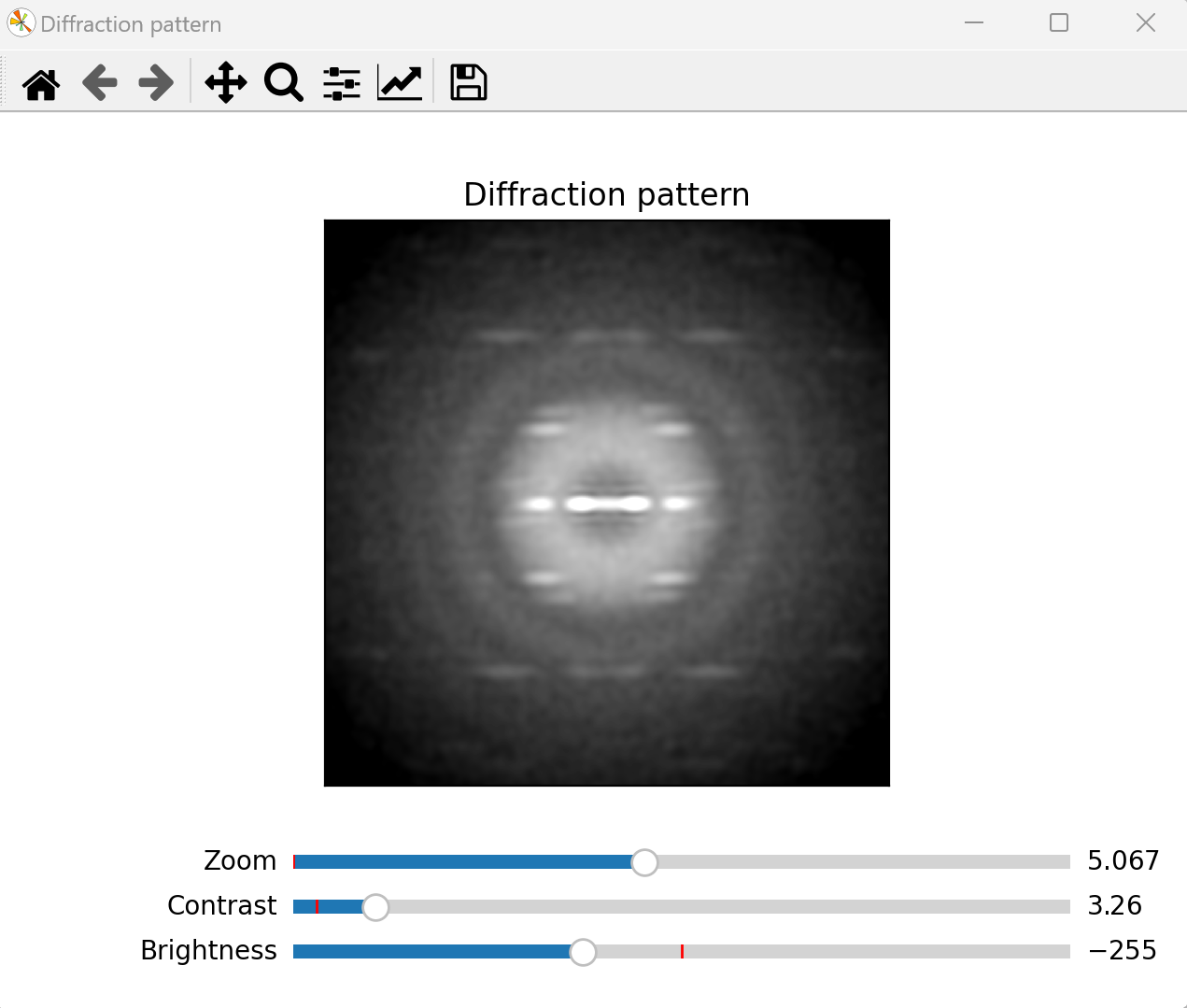
**-Zoom:** This slider can be used to zoom in on the target area, or it can be done by sliding the mouse wheel.

**-Contrast and Brightness:**

The image adjustment calculation formula is as follows:

Image\_adjusted= (Image- mean (Image)) ×Contrast + mean (Image) + Brightness

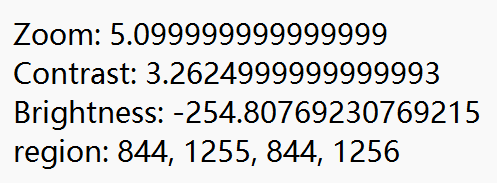
You could remove excess noise by adjusting the Contrast and Brightness sliders to preserve the main diffraction layer lines. In this case, for example, this step could be done like this:



The diffraction pattern

And then, you can just press “space”, then the file named “diffraction\_pattern” could be saved for subsequent indexing, **and a “value” file which contains the information of adjustment was saved at the same time.** You could also get the diffraction pattern of the projection by other software such as EMAN2 and so on. However, in order to facilitate the later screening of data, we recommend use AI-HEAD to finish this process.

\*The “value” file is important for the sorting of the data later.



The “value” file

### Get the information of the diffraction pattern

OK, now that we've finished our preparation, let's get back to our indexing. To achieve indexing of diffraction patterns, it is necessary to know the information of the diffraction center of the diffraction pattern and the information of all effective diffraction points. In fact, the program does not know what a "diffraction pattern" is, nor does it know what information to obtain from the "diffraction pattern". So, AI-HEAD needs to get all of the information by the “Image\_information” module.

There are 2 inputs in this module:

**-FFT image:** You should load your diffraction pattern in this interface.

\*Large screenshots may slow down the computation. The recommended screenshot size is between 400 and 800.

**-lower limit:** You need to enter the lower threshold of the double threshold image segmentation in this interface.

**-upper limit:** You need to enter the upper threshold of the double threshold image segmentation in this interface.

\*Canny algorithm adopts the method of double threshold segmentation for region detection. If the pixel value of a region is less than the minimum threshold, it will not be considered. If the pixel value of an area is greater than the maximum threshold, it is retained. If the gradient value of a certain region is between the high and low threshold, the pixel gradient value is found from the 8 neighborhood of the pixel. If there is a pixel gradient value higher than the high threshold, it is retained; if not, it is not considered.

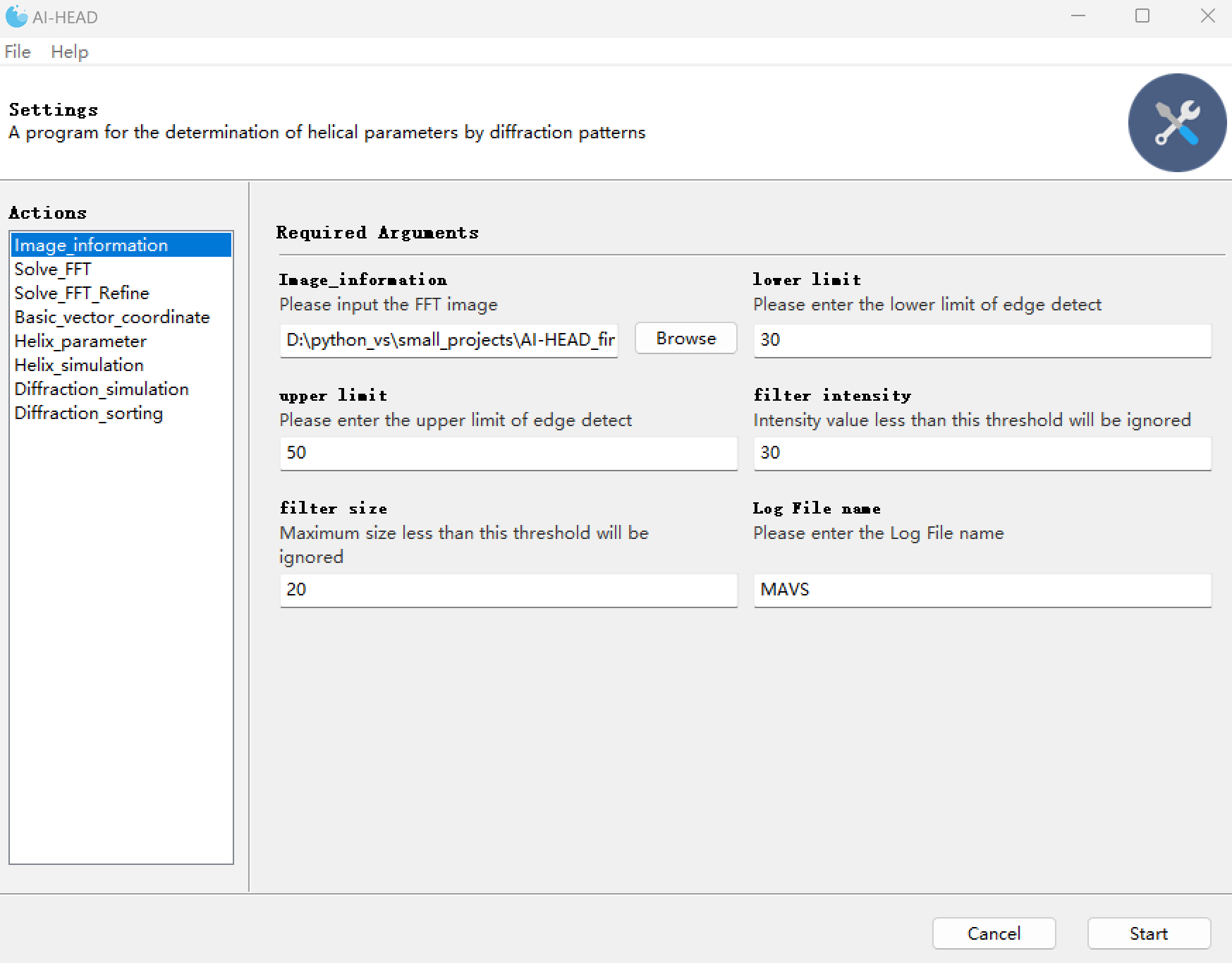
**-filter intensity:** You should enter the intensity threshold in this interface.

\*Regions whose intensity below this threshold will be filtered out.

**-filter size:** You should load your diffraction pattern in this interface.

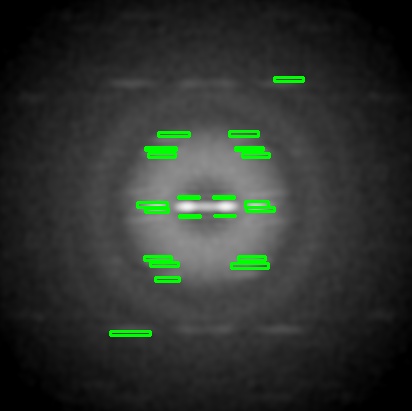
\*Regions whose size below this threshold will be filtered out.

**-Log File name:** This is where you need to enter the name of the log file folder. When you first use this program, it automatically generates a folder named “Image\_information”, and then generate a folder you named under the “Image\_information” folder to record the information. The main information will contain 3 files: the “center” file which contains the center coordinate of the center of diffraction pattern; second, the “region” file which contains all of the range of diffraction region detected; and the edge detection results conducted by Canny algorithm.



Image\_information

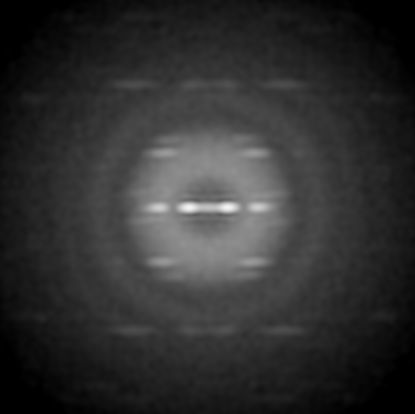
Once you choose the parameters, an interface will pop up to display the results of the edge detection. In this step, Canny algorithm (Canny, 1986) is used to automatically detect the edge of the diffraction pattern and perform the threshold segmentation (N., 1979; A. and S. et al., 2018; D. and H. et al., 2018; Latychevskaia and Fink, 2018; N. and C., 2018; P. and K., 2018). As shown in the figure below, the pink areas are all the actual boundaries detected; The green rectangular area is the actual rectangular area, and the coordinate area of the green area is automatically written to the "region" file. And the result of edge detection will be also saved.



The first interface for displaying the result of edge detection

When you close the second interface, the third interface then pops up to manually supplement areas not detected by the Canny algorithm, or manually segment areas that are not fully segmented

In this case, the diffraction layer lines in the diffraction pattern are very sharp, and the Canny algorithm can find the information of all the diffraction layer lines very accurately, so we could skip this step here. However, when the diffraction pattern is not so clear, the Canny algorithm will not be able to complete the detection and segmentation work well. In that case, manual correction is necessary. You can select undetected areas or areas that are not fully divided by clicking the mouse. These coordinates are then automatically written to the "region" file.



The second interface for manual correction

\*Here, we need to manually pick out the diffraction boxes that Canny did not find

### Auto-indexing of the diffraction pattern by “global searching”

Now that we've got all of the information of diffraction pattern. Next, AI-HEAD will use this information to conduct auto-indexing with a "global searching" strategy by the “Solve\_FFT” module.

There are 6 inputs in this module:

**-FFT image:** You should load your diffraction pattern in this interface.

\*Large screenshots may slow down the computation. The recommended screenshot size is between 400 and 800.

**-The center coordinates:** You should load the “center” file got in “Image\_information” module.

\*The center file should contain only one row of center coordinate data.

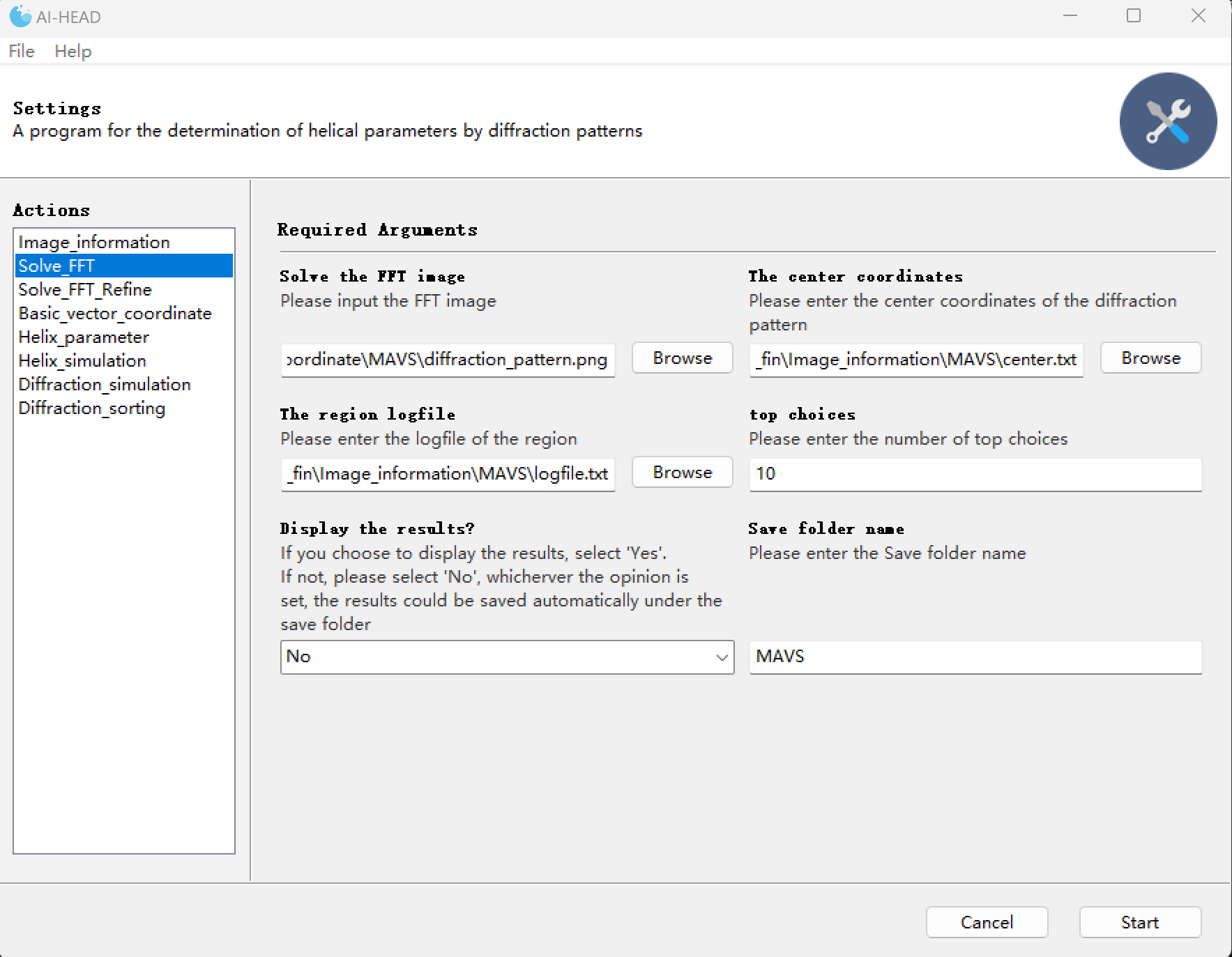
**-The region logfile:** You should load the “region” file got in “Image\_information” module.

**-Top choices:** As AI-HEAD uses a "global searching" strategy, all indexing methods are taken into account. But AI-HEAD will sort all the results according to the conditions of the ideal indexing (such as the number of diffraction layer lines covered under the indexing, the number of diffraction points covered, etc.). Here you can specify the number of optimal AI-HEAD outputs. In most cases, 30 is recommended.

\* In general, indexing at the top of the ranking is ideal. However, as a diffraction pattern can be indexed in many ways, it doesn't mean that results ranking at the bottom is completely wrong. As each pair of basic vectors produces a lattice. When checking the indexing method, the basis vector pairs that produce similar lattices can be considered "equivalent basis vectors". Basic vectors that are farther from the meridian tend to produce multiple sets of possible parameters. So, in summary, when choosing the final indexing method, give priority to the top ones (e.g., top1-5). If several different indexing methods produce similar lattices, then the base vector closer to the meridian and equator is preferred as the final indexing method.

**-Display opinion:** If “Yes” is selected here, the program will bring up a series of screens showing the output of indexing results. No matter what this option selects, the program will save the output in the specified folder.

**-Save folder name:** This is where you need to enter the name of the log file folder. When you first use this program, it automatically generates a folder named “Solve\_FFT”, and then generate a folder you named under the “Solve\_FFT” folder to record the information.



Auto-indexing of the diffraction pattern by “global searching”

After the options are set, the program can be run to conduct “Auto-indexing”. In cases where the diffraction pattern is very clear, “Auto-indexing” is fast and can be completed in less than 1 minute. When the diffraction pattern is very noisy, the speed of indexing is slowed down because the noise is also taken into account in the calculation. In this case, score1 and score2 produce the same lattice, but the position of the basic vectors for score2 is closer to the meridian, which means the number of solution sets calculated by score2 will be smaller, which can reduce the difficulty of subsequent parameter screening. Therefore, we chose score2 as the final indexing method. In this result, different colored dots have different meanings:

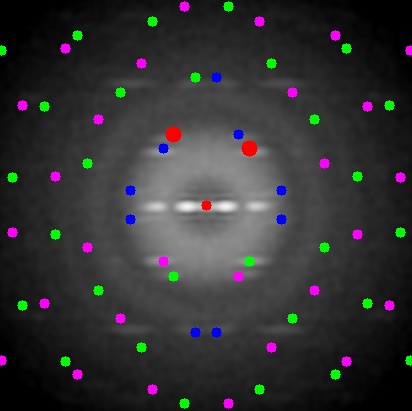
Red point(small): The center of diffraction pattern determined before.

Red points(large): The basic vectors which generate the current lattice.

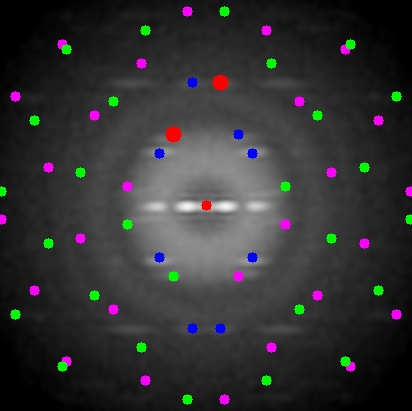
Pink points: The lattice generated by the basic vectors directly.

Green points: The lattice symmetric with the pink lattice about the meridian.

Blue points: Points in lattice or sym-lattice that fall within the diffraction region.



Auto-indexing by “Global seraching” result score 1



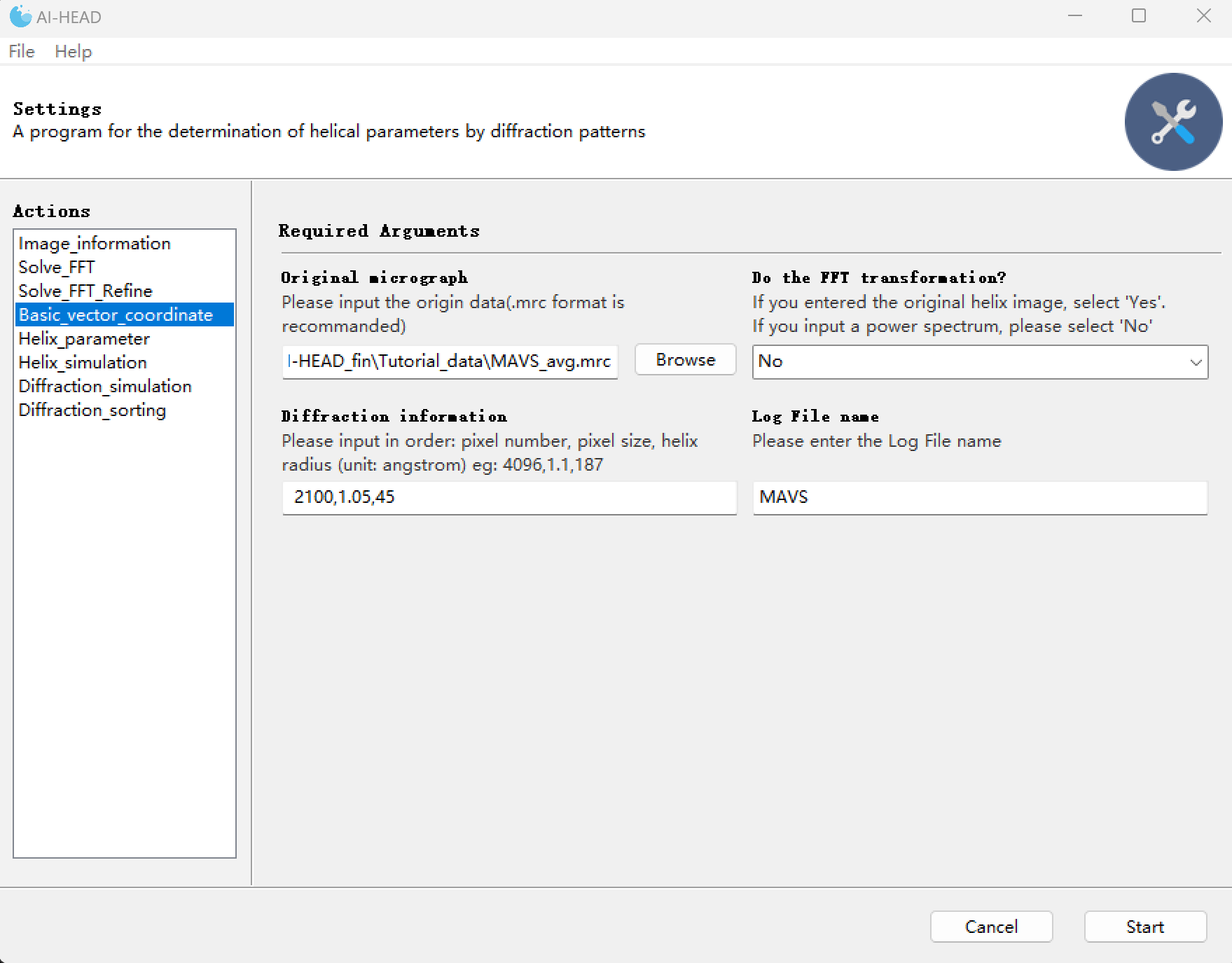
Auto-indexing by “Global seraching” result score 2

In this case, since there are lattice points on each diffraction layer line region, the indexing method selected by the "global searching" strategy can be used, and we can skip the "local refinement" step in “Solve\_FFT\_Refine” module. In most cases, just relying on "global searching" strategy can play a good role. However, as "global searching" takes the center of each diffracted patch as the sampling point, if the diffracted patch is very wide, a large deviation will occur and a very ideal solution cannot be obtained. At that point, "Local refinement" can serve as a complement.

## Calculation of helical parameters

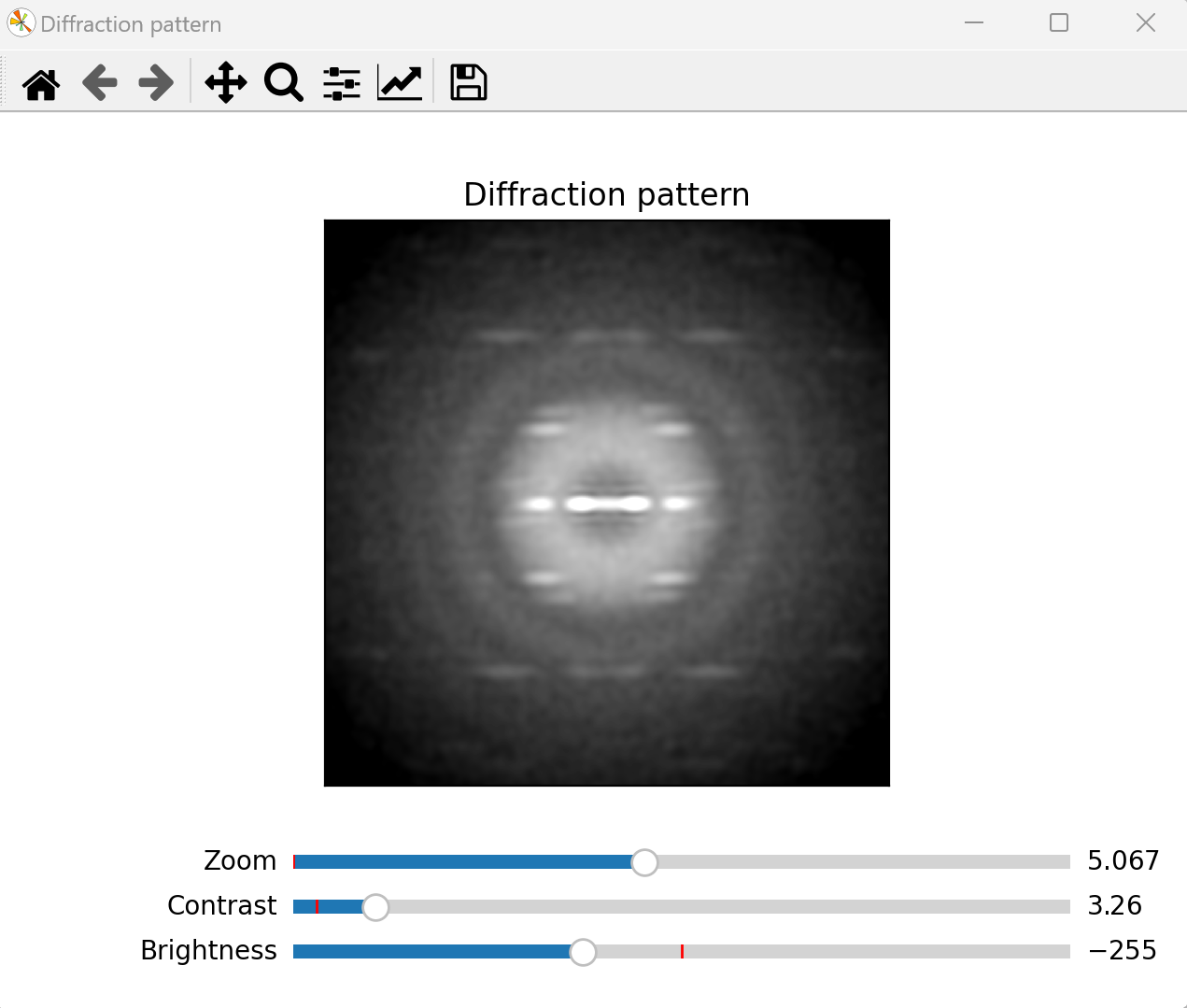
### Determining the “real coordinate” of the basic vectors

Now, AI-HEAD has told you where the basic vectors should be selected for ideal indexing, the indexing process is over, you could calculate the helical parameters based on the indexing results. However, you can't calculate it directly with the coordinates you have. That's because you've zoomed in on the main diffraction area, cut it out and done the whole indexing process. So, we need to go back to the original data and get the real coordinates of the basis vectors. We can still use “Basic\_vector\_coordinate” module to complete this step.



Determining the “real coordinate” of the basic vectors

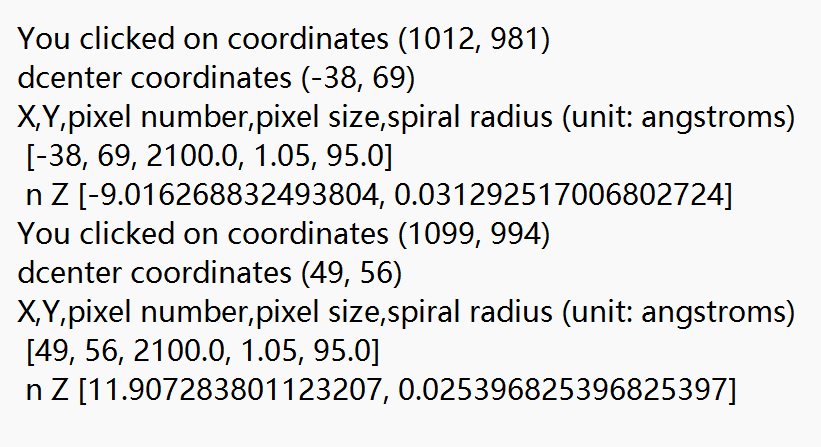
If you have previously obtained a diffraction pattern with AI-HEAD, then all the inputs are consistent with the preparation process. The difference is that we need to double-click in this interface to get the real coordinates of the basic vectors.



Double click here

Double click here

The interface for determining the coordinate of basic vectors



The coordinate will be calculated automatically in terminal

Then, the “real coordinate” of the basic vectors will be automatically recorded in the logfile under the “Basic\_vectors\_coordinate” folder.

### Calculation of helical parameters

Now comes our exciting final step, which is the calculation of the helical parameters. This step could be finished by the “Helix\_parameter” module.

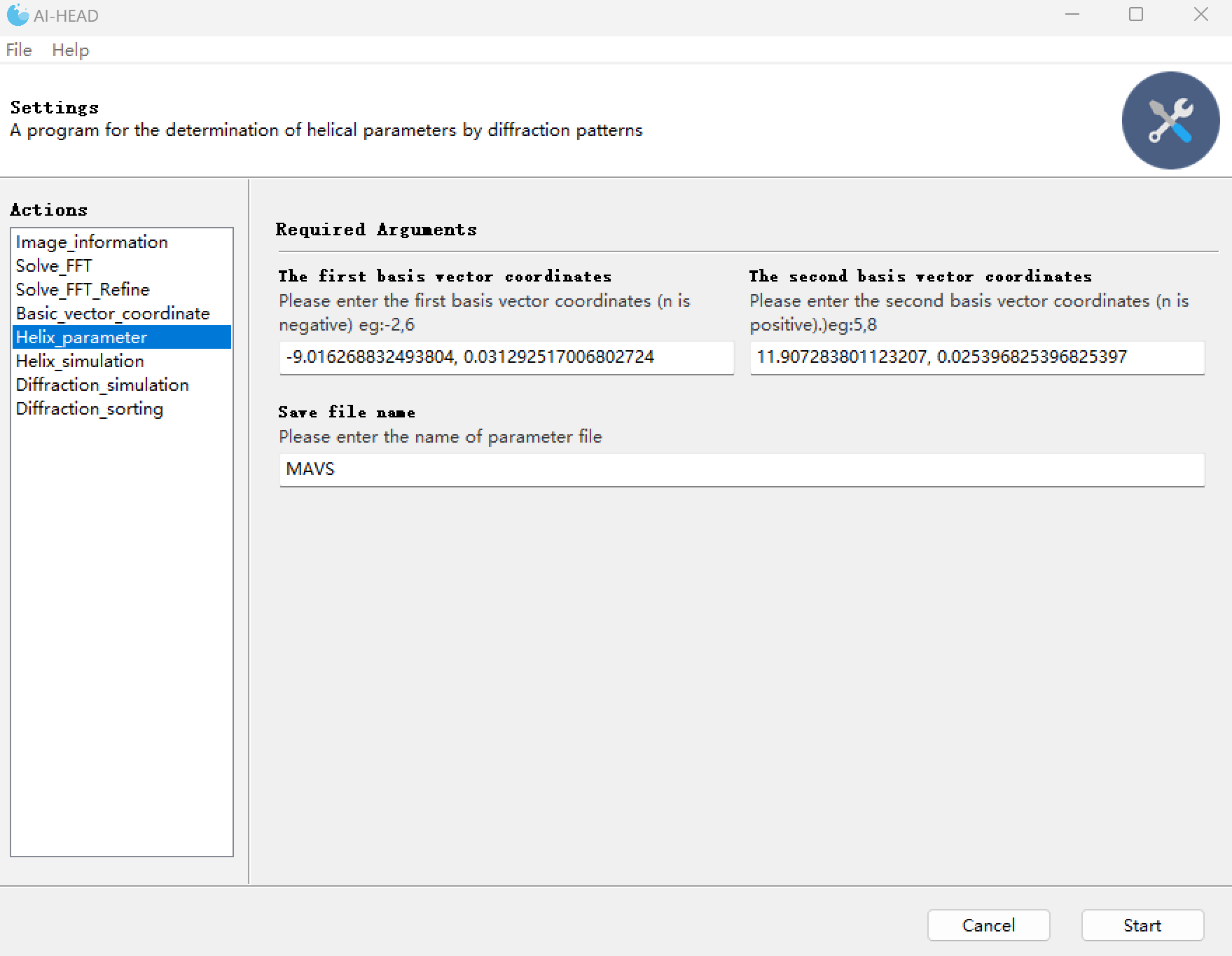
There are 3 inputs in this module:

**-The first basis vector coordinate:** You should input the coordinate got before in “Basic\_vector\_coordinate” module whose n is negative.

**-The second basis vector coordinate:** You should input the coordinate got before in “Basic\_vector\_coordinate” module whose n is positive.

\*The first input box n must be negative, and the second must be positive, otherwise there will be no output.

**-Save file name:** This is where you need to enter the name of the log file. When you first use this program, it automatically generates a folder named “Helix\_parameter”, and then generate two charts you named under the “Helix\_parameter” folder to record the information.



The calculation of helical parameters

Finally, you get a set of solution sets that contain all the possibilities of helical parameters. In this case, we can see that the parameter highlighted in red is very close to the correct parameters. Recorded in EMDB (Rise=5.07Å Twist=-101.2°).

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | [n1,n2] | rise | P | twist | N | k\_value | [Z1,Z2] | [x1\_coff] | [x2\_coff] |
| 0 | [-1, 1] | 17.64 | 31.96 | 198.72 | 1 | [1, 0] | ['0.03129', '0.02540'] | [-1.0, 1.0, 0] | [1.0, -1.0, 1] |
| 1 | [-1, 2] | 11.37 | 31.96 | 128.04 | 1 | [1, 0] | ['0.03129', '0.02540'] | [-1.0, 1.0, 0] | [2.0, -2.0, 1] |
| 2 | [-1, 3] | 8.38 | 31.96 | 94.45 | 1 | [1, 0] | ['0.03129', '0.02540'] | [-1.0, 1.0, 0] | [3.0, -3.0, 1] |
| 3 | [-1, 4] | 6.64 | 31.96 | 74.82 | 1 | [1, 0] | ['0.03129', '0.02540'] | [-1.0, 1.0, 0] | [4.0, -4.0, 1] |
| 4 | [-1, 5] | 5.5 | 31.96 | 61.95 | 1 | [1, 0] | ['0.03129', '0.02540'] | [-1.0, 1.0, 0] | [5.0, -5.0, 1] |
| 5 | [-1, 6] | 4.69 | 31.96 | 52.85 | 1 | [1, 0] | ['0.03129', '0.02540'] | [-1.0, 1.0, 0] | [6.0, -6.0, 1] |
| 6 | [-1, 7] | 4.09 | 31.96 | 46.09 | 1 | [1, 0] | ['0.03129', '0.02540'] | [-1.0, 1.0, 0] | [7.0, -7.0, 1] |
| 7 | [-1, 8] | 3.63 | 31.96 | 40.86 | 1 | [1, 0] | ['0.03129', '0.02540'] | [-1.0, 1.0, 0] | [8.0, -8.0, 1] |
| 8 | [-1, 9] | 3.26 | 31.96 | 36.69 | 1 | [1, 0] | ['0.03129', '0.02540'] | [-1.0, 1.0, 0] | [9.0, -9.0, 1] |
| 9 | [-1, 10] | 2.96 | 31.96 | 33.3 | 1 | [1, 0] | ['0.03129', '0.02540'] | [-1.0, 1.0, 0] | [10.0, -10.0, 1] |
| 10 | [-1, 11] | 2.71 | 31.96 | 30.48 | 1 | [1, 0] | ['0.03129', '0.02540'] | [-1.0, 1.0, 0] | [11.0, -11.0, 1] |
| 11 | [-1, 12] | 2.49 | 31.96 | 28.1 | 1 | [1, 0] | ['0.03129', '0.02540'] | [-1.0, 1.0, 0] | [12.0, -12.0, 1] |
| 12 | [-2, 1] | 12.18 | 39.38 | 111.38 | 1 | [0, 1] | ['0.03129', '0.02540'] | [-2.0, -2.0, 1] | [1.0, 1.0, 0] |
| 13 | [-2, 2] | 17.64 | 63.91 | 99.36 | 2 | [1, 0] | ['0.03129', '0.02540'] | [-1.0, 1.0, 0] | [1.0, -1.0, 1] |
| 14 | [-2, 3] | 6.91 | 17.64 | 141.07 | 1 | [1, 1] | ['0.03129', '0.02540'] | [-2.0, -2.0, 1] | [3.0, 3.0, -1] |
| 15 | [-2, 4] | 11.37 | 63.91 | 64.02 | 2 | [1, 0] | ['0.03129', '0.02540'] | [-1.0, 1.0, 0] | [2.0, -2.0, 1] |
| 16 | [-2, 5] | 4.82 | 11.37 | 152.82 | 1 | [2, 1] | ['0.03129', '0.02540'] | [-2.0, -2.0, 1] | [5.0, 5.0, -2] |
| 17 | [-2, 6] | 8.38 | 63.91 | 47.22 | 2 | [1, 0] | ['0.03129', '0.02540'] | [-1.0, 1.0, 0] | [3.0, -3.0, 1] |
| 18 | [-2, 7] | 3.71 | 8.38 | 159.13 | 1 | [3, 1] | ['0.03129', '0.02540'] | [-2.0, -2.0, 1] | [7.0, 7.0, -3] |
| 19 | [-2, 8] | 6.64 | 63.91 | 37.41 | 2 | [1, 0] | ['0.03129', '0.02540'] | [-1.0, 1.0, 0] | [4.0, -4.0, 1] |
| 20 | [-2, 9] | 3.01 | 6.64 | 163.06 | 1 | [4, 1] | ['0.03129', '0.02540'] | [-2.0, -2.0, 1] | [9.0, 9.0, -4] |
| 21 | [-2, 10] | 5.5 | 63.91 | 30.97 | 2 | [1, 0] | ['0.03129', '0.02540'] | [-1.0, 1.0, 0] | [5.0, -5.0, 1] |
| 22 | [-2, 11] | 2.53 | 5.5 | 165.74 | 1 | [5, 1] | ['0.03129', '0.02540'] | [-2.0, -2.0, 1] | [11.0, 11.0, -5] |
| 23 | [-2, 12] | 4.69 | 63.91 | 26.43 | 2 | [1, 0] | ['0.03129', '0.02540'] | [-1.0, 1.0, 0] | [6.0, -6.0, 1] |
| 24 | [-3, 1] | 9.3 | 39.38 | 85.06 | 1 | [0, 1] | ['0.03129', '0.02540'] | [-3.0, -3.0, 1] | [1.0, 1.0, 0] |
| 25 | [-3, 2] | 7.21 | 17.64 | 147.06 | 1 | [1, 1] | ['0.03129', '0.02540'] | [-3.0, 3.0, -1] | [2.0, -2.0, 1] |
| 26 | [-3, 3] | 17.64 | 95.87 | 66.24 | 3 | [1, 0] | ['0.03129', '0.02540'] | [-1.0, 1.0, 0] | [1.0, -1.0, 1] |
| 27 | [-3, 4] | 4.97 | 17.64 | 101.35 | 1 | [1, 1] | ['0.03129', '0.02540'] | [-3.0, -3.0, 1] | [4.0, 4.0, -1] |
| 28 | [-3, 5] | 4.3 | 11.37 | 136.14 | 1 | [-2, -1] | ['0.03129', '0.02540'] | [-3.0, 3.0, -1] | [5.0, -5.0, 2] |
| 29 | [-3, 6] | 11.37 | 95.87 | 42.68 | 3 | [1, 0] | ['0.03129', '0.02540'] | [-1.0, 1.0, 0] | [2.0, -2.0, 1] |
| 30 | [-3, 7] | 3.39 | 11.37 | 107.28 | 1 | [2, 1] | ['0.03129', '0.02540'] | [-3.0, -3.0, 1] | [7.0, 7.0, -2] |
| 31 | [-3, 8] | 3.06 | 8.38 | 131.5 | 1 | [-3, -1] | ['0.03129', '0.02540'] | [-3.0, 3.0, -1] | [8.0, -8.0, 3] |
| 32 | [-3, 9] | 8.38 | 95.87 | 31.48 | 3 | [1, 0] | ['0.03129', '0.02540'] | [-1.0, 1.0, 0] | [3.0, -3.0, 1] |
| 33 | [-3, 10] | 2.57 | 8.38 | 110.35 | 1 | [3, 1] | ['0.03129', '0.02540'] | [-3.0, -3.0, 1] | [10.0, 10.0, -3] |
| 34 | [-3, 11] | 2.38 | 6.64 | 128.93 | 1 | [-4, -1] | ['0.03129', '0.02540'] | [-3.0, 3.0, -1] | [11.0, -11.0, 4] |
| 35 | [-3, 12] | 6.64 | 95.87 | 24.94 | 3 | [1, 0] | ['0.03129', '0.02540'] | [-1.0, 1.0, 0] | [4.0, -4.0, 1] |
| 36 | [-4, 1] | 7.53 | 39.38 | 68.81 | 1 | [0, 1] | ['0.03129', '0.02540'] | [-4.0, -4.0, 1] | [1.0, 1.0, 0] |
| 37 | [-4, 2] | 12.18 | 78.75 | 55.69 | 2 | [0, 1] | ['0.03129', '0.02540'] | [-2.0, -2.0, 1] | [1.0, 1.0, 0] |
| 38 | [-4, 3] | 5.12 | 17.64 | 104.41 | 1 | [1, 1] | ['0.03129', '0.02540'] | [-4.0, 4.0, -1] | [3.0, -3.0, 1] |
| 39 | [-4, 4] | 17.64 | 127.83 | 49.68 | 4 | [1, 0] | ['0.03129', '0.02540'] | [-1.0, 1.0, 0] | [1.0, -1.0, 1] |
| 40 | [-4, 5] | 3.88 | 17.64 | 79.09 | 1 | [1, 1] | ['0.03129', '0.02540'] | [-4.0, -4.0, 1] | [5.0, 5.0, -1] |
| 41 | [-4, 6] | 6.91 | 35.28 | 70.53 | 2 | [1, 1] | ['0.03129', '0.02540'] | [-2.0, -2.0, 1] | [3.0, 3.0, -1] |
| 42 | [-4, 7] | 3.12 | 11.37 | 98.78 | 1 | [-2, -1] | ['0.03129', '0.02540'] | [-4.0, 4.0, -1] | [7.0, -7.0, 2] |
| 43 | [-4, 8] | 11.37 | 127.83 | 32.01 | 4 | [1, 0] | ['0.03129', '0.02540'] | [-1.0, 1.0, 0] | [2.0, -2.0, 1] |
| 44 | [-4, 9] | 2.61 | 11.37 | 82.65 | 1 | [2, 1] | ['0.03129', '0.02540'] | [-4.0, -4.0, 1] | [9.0, 9.0, -2] |
| 45 | [-4, 10] | 4.82 | 22.73 | 76.41 | 2 | [2, 1] | ['0.03129', '0.02540'] | [-2.0, -2.0, 1] | [5.0, 5.0, -2] |
| 46 | [-4, 11] | 2.24 | 8.38 | 96.32 | 1 | [-3, -1] | ['0.03129', '0.02540'] | [-4.0, 4.0, -1] | [11.0, -11.0, 3] |
| 47 | [-4, 12] | 8.38 | 127.83 | 23.61 | 4 | [1, 0] | ['0.03129', '0.02540'] | [-1.0, 1.0, 0] | [3.0, -3.0, 1] |
| 48 | [-5, 1] | 6.32 | 39.38 | 57.77 | 1 | [0, 1] | ['0.03129', '0.02540'] | [-5.0, -5.0, 1] | [1.0, 1.0, 0] |
| 49 | [-5, 2] | 5.28 | 12.18 | 155.89 | 1 | [1, 2] | ['0.03129', '0.02540'] | [-5.0, 5.0, -2] | [2.0, -2.0, 1] |
| 50 | [-5, 3] | 4.53 | 12.18 | 133.8 | 1 | [-1, -2] | ['0.03129', '0.02540'] | [-5.0, -5.0, 2] | [3.0, 3.0, -1] |
| 51 | [-5, 4] | 3.97 | 17.64 | 80.94 | 1 | [1, 1] | ['0.03129', '0.02540'] | [-5.0, 5.0, -1] | [4.0, -4.0, 1] |
| 52 | [-5, 5] | 17.64 | 159.78 | 39.74 | 5 | [1, 0] | ['0.03129', '0.02540'] | [-1.0, 1.0, 0] | [1.0, -1.0, 1] |
| 53 | [-5, 6] | 3.18 | 17.64 | 64.84 | 1 | [1, 1] | ['0.03129', '0.02540'] | [-5.0, -5.0, 1] | [6.0, 6.0, -1] |
| 54 | [-5, 7] | 2.89 | 6.91 | 150.51 | 1 | [-3, -2] | ['0.03129', '0.02540'] | [-5.0, 5.0, -2] | [7.0, -7.0, 3] |
| 55 | [-5, 8] | 2.65 | 6.91 | 138.03 | 1 | [3, 2] | ['0.03129', '0.02540'] | [-5.0, -5.0, 2] | [8.0, 8.0, -3] |
| 56 | [-5, 9] | 2.45 | 11.37 | 77.51 | 1 | [-2, -1] | ['0.03129', '0.02540'] | [-5.0, 5.0, -1] | [9.0, -9.0, 2] |
| 57 | [-5, 10] | 11.37 | 159.78 | 25.61 | 5 | [1, 0] | ['0.03129', '0.02540'] | [-1.0, 1.0, 0] | [2.0, -2.0, 1] |
| 58 | [-5, 11] | 2.12 | 11.37 | 67.22 | 1 | [2, 1] | ['0.03129', '0.02540'] | [-5.0, -5.0, 1] | [11.0, 11.0, -2] |
| 59 | [-5, 12] | 1.99 | 4.82 | 148.48 | 1 | [-5, -2] | ['0.03129', '0.02540'] | [-5.0, 5.0, -2] | [12.0, -12.0, 5] |
| 60 | [-6, 1] | 5.44 | 39.38 | 49.78 | 1 | [0, 1] | ['0.03129', '0.02540'] | [-6.0, -6.0, 1] | [1.0, 1.0, 0] |
| 61 | [-6, 2] | 9.3 | 78.75 | 42.53 | 2 | [0, 1] | ['0.03129', '0.02540'] | [-3.0, -3.0, 1] | [1.0, 1.0, 0] |
| 62 | [-6, 3] | 12.18 | 118.12 | 37.13 | 3 | [0, 1] | ['0.03129', '0.02540'] | [-2.0, -2.0, 1] | [1.0, 1.0, 0] |
| 63 | [-6, 4] | 7.21 | 35.28 | 73.53 | 2 | [1, 1] | ['0.03129', '0.02540'] | [-3.0, 3.0, -1] | [2.0, -2.0, 1] |
| 64 | [-6, 5] | 3.24 | 17.64 | 66.08 | 1 | [1, 1] | ['0.03129', '0.02540'] | [-6.0, 6.0, -1] | [5.0, -5.0, 1] |
| 65 | [-6, 6] | 17.64 | 191.74 | 33.12 | 6 | [1, 0] | ['0.03129', '0.02540'] | [-1.0, 1.0, 0] | [1.0, -1.0, 1] |
| 66 | [-6, 7] | 2.69 | 17.64 | 54.95 | 1 | [1, 1] | ['0.03129', '0.02540'] | [-6.0, -6.0, 1] | [7.0, 7.0, -1] |
| 67 | [-6, 8] | 4.97 | 35.28 | 50.68 | 2 | [1, 1] | ['0.03129', '0.02540'] | [-3.0, -3.0, 1] | [4.0, 4.0, -1] |
| 68 | [-6, 9] | 6.91 | 52.92 | 47.02 | 3 | [1, 1] | ['0.03129', '0.02540'] | [-2.0, -2.0, 1] | [3.0, 3.0, -1] |
| 69 | [-6, 10] | 4.3 | 22.73 | 68.07 | 2 | [-2, -1] | ['0.03129', '0.02540'] | [-3.0, 3.0, -1] | [5.0, -5.0, 2] |
| 70 | [-6, 11] | 2.01 | 11.37 | 63.78 | 1 | [-2, -1] | ['0.03129', '0.02540'] | [-6.0, 6.0, -1] | [11.0, -11.0, 2] |
| 71 | [-6, 12] | 11.37 | 191.74 | 21.34 | 6 | [1, 0] | ['0.03129', '0.02540'] | [-1.0, 1.0, 0] | [2.0, -2.0, 1] |
| 72 | [-7, 1] | 4.78 | 39.38 | 43.73 | 1 | [0, 1] | ['0.03129', '0.02540'] | [-7.0, -7.0, 1] | [1.0, 1.0, 0] |
| 73 | [-7, 2] | 4.16 | 9.3 | 160.98 | 1 | [1, 3] | ['0.03129', '0.02540'] | [-7.0, 7.0, -3] | [2.0, -2.0, 1] |
| 74 | [-7, 3] | 3.68 | 12.18 | 108.78 | 1 | [1, 2] | ['0.03129', '0.02540'] | [-7.0, 7.0, -2] | [3.0, -3.0, 1] |
| 75 | [-7, 4] | 3.3 | 12.18 | 97.54 | 1 | [-1, -2] | ['0.03129', '0.02540'] | [-7.0, -7.0, 2] | [4.0, 4.0, -1] |
| 76 | [-7, 5] | 2.99 | 7.21 | 149.47 | 1 | [-2, -3] | ['0.03129', '0.02540'] | [-7.0, -7.0, 3] | [5.0, 5.0, -2] |
| 77 | [-7, 6] | 2.74 | 17.64 | 55.83 | 1 | [1, 1] | ['0.03129', '0.02540'] | [-7.0, 7.0, -1] | [6.0, -6.0, 1] |
| 78 | [-7, 7] | 17.64 | 223.7 | 28.39 | 7 | [1, 0] | ['0.03129', '0.02540'] | [-1.0, 1.0, 0] | [1.0, -1.0, 1] |
| 79 | [-7, 8] | 2.34 | 17.64 | 47.67 | 1 | [1, 1] | ['0.03129', '0.02540'] | [-7.0, -7.0, 1] | [8.0, 8.0, -1] |
| 80 | [-7, 9] | 2.18 | 4.97 | 157.79 | 1 | [-4, -3] | ['0.03129', '0.02540'] | [-7.0, 7.0, -3] | [9.0, -9.0, 4] |
| 81 | [-7, 10] | 2.04 | 6.91 | 106.14 | 1 | [-3, -2] | ['0.03129', '0.02540'] | [-7.0, 7.0, -2] | [10.0, -10.0, 3] |
| 82 | [-7, 11] | 1.92 | 6.91 | 99.77 | 1 | [3, 2] | ['0.03129', '0.02540'] | [-7.0, -7.0, 2] | [11.0, 11.0, -3] |
| 83 | [-7, 12] | 1.81 | 4.3 | 151.38 | 1 | [5, 3] | ['0.03129', '0.02540'] | [-7.0, -7.0, 3] | [12.0, 12.0, -5] |
| 84 | [-8, 1] | 4.26 | 39.38 | 38.99 | 1 | [0, 1] | ['0.03129', '0.02540'] | [-8.0, -8.0, 1] | [1.0, 1.0, 0] |
| 85 | [-8, 2] | 7.53 | 78.75 | 34.4 | 2 | [0, 1] | ['0.03129', '0.02540'] | [-4.0, -4.0, 1] | [1.0, 1.0, 0] |
| 86 | [-8, 3] | 3.37 | 9.3 | 130.26 | 1 | [-1, -3] | ['0.03129', '0.02540'] | [-8.0, -8.0, 3] | [3.0, 3.0, -1] |
| 87 | [-8, 4] | 12.18 | 157.5 | 27.85 | 4 | [0, 1] | ['0.03129', '0.02540'] | [-2.0, -2.0, 1] | [1.0, 1.0, 0] |
| 88 | [-8, 5] | 2.78 | 7.21 | 138.92 | 1 | [2, 3] | ['0.03129', '0.02540'] | [-8.0, 8.0, -3] | [5.0, -5.0, 2] |
| 89 | [-8, 6] | 5.12 | 35.28 | 52.2 | 2 | [1, 1] | ['0.03129', '0.02540'] | [-4.0, 4.0, -1] | [3.0, -3.0, 1] |
| 90 | [-8, 7] | 2.37 | 17.64 | 48.34 | 1 | [1, 1] | ['0.03129', '0.02540'] | [-8.0, 8.0, -1] | [7.0, -7.0, 1] |
| 91 | [-8, 8] | 17.64 | 255.65 | 24.84 | 8 | [1, 0] | ['0.03129', '0.02540'] | [-1.0, 1.0, 0] | [1.0, -1.0, 1] |
| 92 | [-8, 9] | 2.06 | 17.64 | 42.1 | 1 | [1, 1] | ['0.03129', '0.02540'] | [-8.0, -8.0, 1] | [9.0, 9.0, -1] |
| 93 | [-8, 10] | 3.88 | 35.28 | 39.54 | 2 | [1, 1] | ['0.03129', '0.02540'] | [-4.0, -4.0, 1] | [5.0, 5.0, -1] |
| 94 | [-8, 11] | 1.83 | 4.97 | 132.43 | 1 | [4, 3] | ['0.03129', '0.02540'] | [-8.0, -8.0, 3] | [11.0, 11.0, -4] |
| 95 | [-8, 12] | 6.91 | 70.56 | 35.27 | 4 | [1, 1] | ['0.03129', '0.02540'] | [-2.0, -2.0, 1] | [3.0, 3.0, -1] |
| 96 | [-9, 1] | 3.85 | 39.38 | 35.18 | 1 | [0, 1] | ['0.03129', '0.02540'] | [-9.0, -9.0, 1] | [1.0, 1.0, 0] |
| 97 | [-9, 2] | 3.43 | 7.53 | 164.3 | 1 | [1, 4] | ['0.03129', '0.02540'] | [-9.0, 9.0, -4] | [2.0, -2.0, 1] |
| 98 | [-9, 3] | 9.3 | 118.12 | 28.35 | 3 | [0, 1] | ['0.03129', '0.02540'] | [-3.0, -3.0, 1] | [1.0, 1.0, 0] |
| 99 | [-9, 4] | 2.83 | 12.18 | 83.54 | 1 | [1, 2] | ['0.03129', '0.02540'] | [-9.0, 9.0, -2] | [4.0, -4.0, 1] |
| 100 | [-9, 5] | 2.6 | 12.18 | 76.75 | 1 | [-1, -2] | ['0.03129', '0.02540'] | [-9.0, -9.0, 2] | [5.0, 5.0, -1] |
| 101 | [-9, 6] | 7.21 | 52.92 | 49.02 | 3 | [1, 1] | ['0.03129', '0.02540'] | [-3.0, 3.0, -1] | [2.0, -2.0, 1] |
| 102 | [-9, 7] | 2.23 | 5.12 | 157.2 | 1 | [-3, -4] | ['0.03129', '0.02540'] | [-9.0, -9.0, 4] | [7.0, 7.0, -3] |
| 103 | [-9, 8] | 2.09 | 17.64 | 42.61 | 1 | [1, 1] | ['0.03129', '0.02540'] | [-9.0, 9.0, -1] | [8.0, -8.0, 1] |
| 104 | [-9, 9] | 17.64 | 287.61 | 22.08 | 9 | [1, 0] | ['0.03129', '0.02540'] | [-1.0, 1.0, 0] | [1.0, -1.0, 1] |
| 105 | [-9, 10] | 1.85 | 17.64 | 37.69 | 1 | [1, 1] | ['0.03129', '0.02540'] | [-9.0, -9.0, 1] | [10.0, 10.0, -1] |
| 106 | [-9, 11] | 1.75 | 3.88 | 162.19 | 1 | [-5, -4] | ['0.03129', '0.02540'] | [-9.0, 9.0, -4] | [11.0, -11.0, 5] |
| 107 | [-9, 12] | 4.97 | 52.92 | 33.78 | 3 | [1, 1] | ['0.03129', '0.02540'] | [-3.0, -3.0, 1] | [4.0, 4.0, -1] |
| 108 | [-10, 1] | 3.51 | 39.38 | 32.05 | 1 | [0, 1] | ['0.03129', '0.02540'] | [-10.0, -10.0, 1] | [1.0, 1.0, 0] |
| 109 | [-10, 2] | 6.32 | 78.75 | 28.88 | 2 | [0, 1] | ['0.03129', '0.02540'] | [-5.0, -5.0, 1] | [1.0, 1.0, 0] |
| 110 | [-10, 3] | 2.87 | 9.3 | 111.24 | 1 | [1, 3] | ['0.03129', '0.02540'] | [-10.0, 10.0, -3] | [3.0, -3.0, 1] |
| 111 | [-10, 4] | 5.28 | 24.36 | 77.94 | 2 | [1, 2] | ['0.03129', '0.02540'] | [-5.0, 5.0, -2] | [2.0, -2.0, 1] |
| 112 | [-10, 5] | 12.18 | 196.88 | 22.28 | 5 | [0, 1] | ['0.03129', '0.02540'] | [-2.0, -2.0, 1] | [1.0, 1.0, 0] |
| 113 | [-10, 6] | 4.53 | 24.36 | 66.9 | 2 | [-1, -2] | ['0.03129', '0.02540'] | [-5.0, -5.0, 2] | [3.0, 3.0, -1] |
| 114 | [-10, 7] | 2.11 | 7.21 | 105.62 | 1 | [-2, -3] | ['0.03129', '0.02540'] | [-10.0, -10.0, 3] | [7.0, 7.0, -2] |
| 115 | [-10, 8] | 3.97 | 35.28 | 40.47 | 2 | [1, 1] | ['0.03129', '0.02540'] | [-5.0, 5.0, -1] | [4.0, -4.0, 1] |
| 116 | [-10, 9] | 1.87 | 17.64 | 38.1 | 1 | [1, 1] | ['0.03129', '0.02540'] | [-10.0, 10.0, -1] | [9.0, -9.0, 1] |
| 117 | [-10, 10] | 17.64 | 319.57 | 19.87 | 10 | [1, 0] | ['0.03129', '0.02540'] | [-1.0, 1.0, 0] | [1.0, -1.0, 1] |
| 118 | [-10, 11] | 1.67 | 17.64 | 34.12 | 1 | [1, 1] | ['0.03129', '0.02540'] | [-10.0, -10.0, 1] | [11.0, 11.0, -1] |
| 119 | [-10, 12] | 3.18 | 35.28 | 32.42 | 2 | [1, 1] | ['0.03129', '0.02540'] | [-5.0, -5.0, 1] | [6.0, 6.0, -1] |

The main chart of results

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | [n1,n2] | rise | P | twist | N | k\_value | [Z1,Z2] | [x1\_coff] | [x2\_coff] |
| 0 | [-1, 1] | 17.64 | 39.38 | 161.28 | 1 | [0, 1] | ['0.03129', '0.02540'] | [-1.0, -1.0, 1] | [1.0, 1.0, 0] |
| 1 | [-2, 2] | 17.64 | 78.75 | 80.64 | 2 | [0, 1] | ['0.03129', '0.02540'] | [-1.0, -1.0, 1] | [1.0, 1.0, 0] |
| 2 | [-3, 3] | 17.64 | 118.12 | 53.76 | 3 | [0, 1] | ['0.03129', '0.02540'] | [-1.0, -1.0, 1] | [1.0, 1.0, 0] |
| 3 | [-4, 4] | 17.64 | 157.5 | 40.32 | 4 | [0, 1] | ['0.03129', '0.02540'] | [-1.0, -1.0, 1] | [1.0, 1.0, 0] |
| 4 | [-5, 5] | 17.64 | 196.88 | 32.26 | 5 | [0, 1] | ['0.03129', '0.02540'] | [-1.0, -1.0, 1] | [1.0, 1.0, 0] |
| 5 | [-6, 6] | 17.64 | 236.25 | 26.88 | 6 | [0, 1] | ['0.03129', '0.02540'] | [-1.0, -1.0, 1] | [1.0, 1.0, 0] |
| 6 | [-7, 7] | 17.64 | 275.62 | 23.04 | 7 | [0, 1] | ['0.03129', '0.02540'] | [-1.0, -1.0, 1] | [1.0, 1.0, 0] |
| 7 | [-8, 8] | 17.64 | 315 | 20.16 | 8 | [0, 1] | ['0.03129', '0.02540'] | [-1.0, -1.0, 1] | [1.0, 1.0, 0] |
| 8 | [-9, 9] | 17.64 | 354.38 | 17.92 | 9 | [0, 1] | ['0.03129', '0.02540'] | [-1.0, -1.0, 1] | [1.0, 1.0, 0] |
| 9 | [-10, 10] | 17.64 | 393.75 | 16.13 | 10 | [0, 1] | ['0.03129', '0.02540'] | [-1.0, -1.0, 1] | [1.0, 1.0, 0] |

The additional chart of results