**Terahertz Thermal Emission Optimization**

**with Genetic Algorithm**

Veronika Strnadovà, University of New Mexico

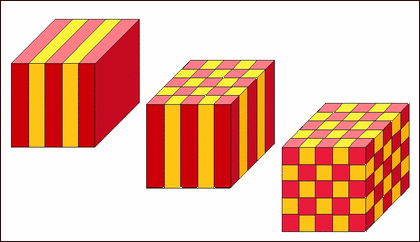
Casey Irvine, University of Arizona

Natalie Coston, Northern Arizona University

**Photonic Crystal Overview**

Photonic band gap structures, or photonic crystals, have attracted attention in the past few years for their ability to control and manipulate the flow of light. They have applications in atomic physics, optics, microwave devices, vacuum electronic devices, and particle accelerators. They are composed of periodic internal regions that vary by the dielectric constant of the material. The dielectric material they are composed of can vary in one-dimension, two-dimensions, and three dimensions, as shown in the figure below.

1-D



2-D

3-D

In March of 2006 Dr. Samokhvalova *et al.* published a paper that analyzed the dispersion relations of two-dimensional photonic crystals and from them determined the dispersion characteristics. Prior to this paper, only the dispersion characteristics for a photonic crystal varying in one-dimension, known as a Bragg Structure, were known. Later in August 2008, Dr. Kano et al. published a paper in which they derived an analytic formula for the computed energy density of a three-dimensional photonic crystal. However, the computations needed to optimize emission for a three dimensional structure require more computing power than was available to this research.

In our research we looked at the optimal construction for a two-dimensional varying photonic crystal that would radiate in a non-plank manner. In particular, we wanted to optimize the amount of energy emitted by a photonic crystal in the terahertz regime. We examined a crystal with the following structure.

This structure has six variables: are the dielectric constants of each material used in the two-dimensional block crystal, and and show how much of each material is used. One of the restrictions on the two-dimensional structure, as shown by Samokhvalova *et al.,* is that epsilon1 + epsilon 3 = epsilon 2 + epsilon 4 (or ). Kano *et al.* were able to accurately calculate the density of states for a two-dimensional crystal by integrating the analytic dispersion relation. Kano *et al.* derived the gradient and arc length along fixed frequency isocontours of the dispersion relation surface and integrated in the irreducible Brillouin zone. Taking the analytic solution for the density of states, Kano *et al.* then use it to analyze terahertz radiation from a thermally pumped photonic crystal. We are interested in the terahertz range because we are comparing the photonic crystal to a standard two-dimensional black body with temperature 310.15 K radiating into free space; the maximum of the spectral energy density for this black body is in the terahertz range. As Kano et al. show, the spectral energy density of the crystal, computed by multiplying the density of states with average photon energy, deviates significantly from Planck behavior at large frequencies. We experimented with the structure of the crystal to obtain an optimum spectral energy density, given some frequency range.

**The Genetic Algorithm**

Since our research was primarily an optimization problem, much of it focused on the algorithm we used to numerically find a maximum value for the thermal emission of the two-dimensional photonic crystal in a given frequency range. We used the genetic algorithm, essentially a search technique that starts with a random sample of sets of variables (vectors in Matlab) and converges toward a global optimum. It was derived to solve optimization problems based on natural selection, the process that drives biological evolution. Because the genetic algorithm is a stochastic search engine, we cannot guarantee that an optimal solution will be reached after a fixed number of iterations. However, it has been tested against several other optimization methods and, as shown by Greenhalgh and Marshall, running the genetic algorithm for long enough can “guarantee convergence to the global optimum with any prescribed level of confidence.”

The function that we needed to optimize was very rough and had many local maxima which made it difficult to calculate the true optimal solution. To find the global maxima, the genetic algorithm (GA) repeatedly modifies a population of individual “points.”In our case, each distinct combination of the six variables gives a single point in the population. Over a number of generations, these solutions “evolve” towards the optimal solution, i.e. the best combination of.

The GA starts with an initial population made up of individual population points. The genetic algorithm is given a number, say, 100 of these points to test. It tests their “fitness value” (given by the fitness function, see below). It then compares them and ranks them. The GA will then do one of three things to each point in the population. It will either keep a point for the next generation (selection), combine two points to get a new point (crossover), or randomly change the point completely (mutation). These processes can be visualized with these diagrams.

Crossover

Selection

Mutation

These new points go back into the GA as the next generation. This process is repeated until one of the parameters of the algorithm stops the process. The process can be stopped if all the population points are close enough together for the “tolerance” to be satisfied, the number of allowed generations is exceeded, or if the best value does not change over a certain number of generations (stall generations).The value the GA estimates the optimal area under the computed energy density curve. This value corresponds to a specific combination of our six variables ().

**The Fitness Function**

A crucial part of the genetic algorithm is the fitness function, which determines the fitness value of each individual in the population. The GA ranks individuals based on their fitness values, then performs either selection, crossover, or mutation, depending on the individuals' rank. In other words, we need to find a way to rank the population of vectors whose elements contain the values of a0, b0, ε1, ε2, ε3, and ε4. Since we are trying to find the maximum energy density within a given frequency range, we fix the frequency range that we input to the genetic algorithm and rank individuals according to the value of the integral of the energy density in that range. The fitness value is the value of this integral – a greater integral value gives a better fitness value. The fitness function we use is:

where CED is computed energy density, and ω1 through ω2 is our frequency range. Note that we had to negate the fitness function since the GA only finds minima. We also scaled the frequency to (chi), such that, where *c* is the speed of light in a vacuum and *a* is the length scale.

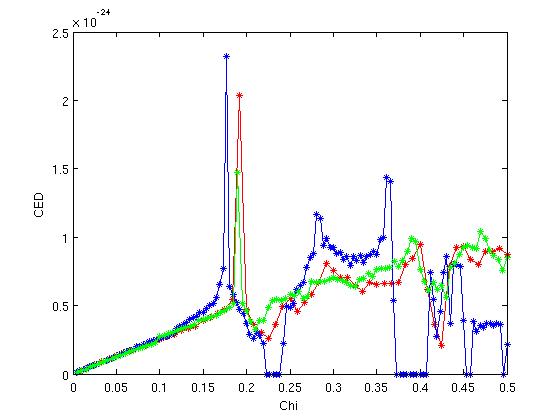
**Mutation Function**

The mutation function determines which “points” in the population, or which parts of which points, will be mutated and how they will be mutated. Mutation adds randomness to the algorithm, allowing it to explore options that crossover alone may not encounter. In the *ga* function found in Matlab, mutation can be achieved using either an adapt-feasible, uniform, or gaussian mutation function. The adapt-feasible function randomly generates directions that are adaptive with respect to the last successful or unsuccessful generation. We used the adapt-feasible function for most of our *ga* runs. The uniform function selects a fraction of the vector entries of an individual for mutation, where each entry has a probability of being mutated. Then each selected entry is replaced by a random number selected uniformly from the range for that entry. The Gaussian function adds a random number taken from a Gaussian distribution with mean 0 to each entry of the vector being mutated.

We spent some time testing the mutation function, but not enough to give conclusive results. We will discuss this later on in the paper.

**Varying the Genetic Algorithm**

The method that the genetic algorithm uses to evaluate the optimal solution does not change. However, there were a variety of aspects of the GA that we could alter in order to achieve an optimal solution more efficiently. One of the aspects that we experimented with was “Nchi”. This is the number of sample points used to integrate over an interval of χ’s. The higher value for Nchi we use the more precise our integration is. We ran runs through the GA changing only the value of Nchi and compared the results. The following figures show the data we collected.



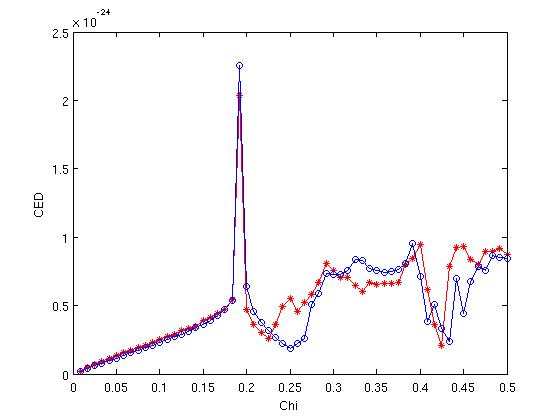
Red = 60 Nchi

Green = 100 Nchi

Blue = 130 Nchi

After analyzing our data we found that higher Nchi does give a better solution, but somewhere between 60 and 100 is best. This gives an accurate integration with a reasonable run time.

We also tried adjusting the tolerance of our GA. The tolerance specifies how close together the points must be for the GA to terminate. We examined the graphs and data and found that although a higher tolerance gives a smoother graph, it is not worth running. When the tolerance is higher the run times increase dramatically for results not much better than ones with a lower tolerance. This can be seen in the figures below.



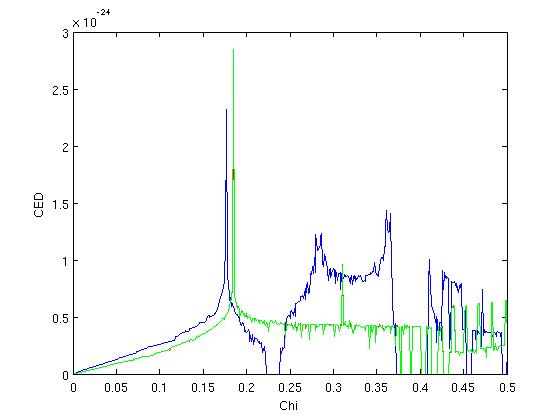
Red = lower tolerance, 17 hours to satisfy algorithm

Blue = higher tolerance, 46 hours to satisfy algorithm

Another effect that raising the tolerance had on our runs of the GA was sometimes the GA would not converge and would continue to run until the maximum generations was exceeded. This was not desired because it did not guarantee settling on a minimum of some sort. Due to these effects, we used values of as the tolerance for the adaptive integration of the dispersion relation, as the step size in the x-direction for the dispersion relation, and as step size in the y-direction for the dispersion relation.

In our runs of the genetic algorithm we experimented with population size. This is the number of population points the GA looks at in each generation. We tried runs using 20 points, 40 points, 60 points, 80 points, and 100 points. We found that it more effective to use a population of around 60 and have a high number of generations. This runs faster than a high population size because the GA does not have to analyze as many population points at each generation. Therefore, in most of our runs we used a population size of about 60 points.

Another parameter we varied in our GA is called initial population. Here, we were able to specify a population point in the initial population for the GA to test. This became useful as we wanted to test our minima to see if they could get better. We tried using the minimum from a run and plugging it in as an initial population point in the next run. We found that this consistently improved our minimum value, as shown by the table and graph below.



|  |  |  |  |
| --- | --- | --- | --- |
|  | **Blue** | **Red** | **Green** |
| **Min** | -0.264724 | -0.308823 | -0.313794 |
| **Ao** | 0.23272 | 0.238085 | 0.238085 |
| **Bo** | 0.227256 | 0.24172 | 0.247172 |
| **ε1** | 19.219 | 14.0159 | 14.0159 |
| **ε2** | 10.2689 | 1.18275 | 1.18277 |
| **ε3** | 1.71816 | 1.00561 | 1.00559 |
| **ε4** | 10.6683 | 13.8387 | 13.8387 |

In the case of this data, I used the variables that the blue graph returned and plugged them in as a population point and ran the GA again. This returned the minimum shown in the red data (the red graph is laying almost directly behind the green graph). The minimum is better than in the blue data. We then used this minimum as a population point, ran the GA, and got back the green data. This minimum is once again better. We tested this repeatedly with other runs and found similar results. We concluded that giving the genetic algorithm an initial population point that is a possible optimal solution improves the result.

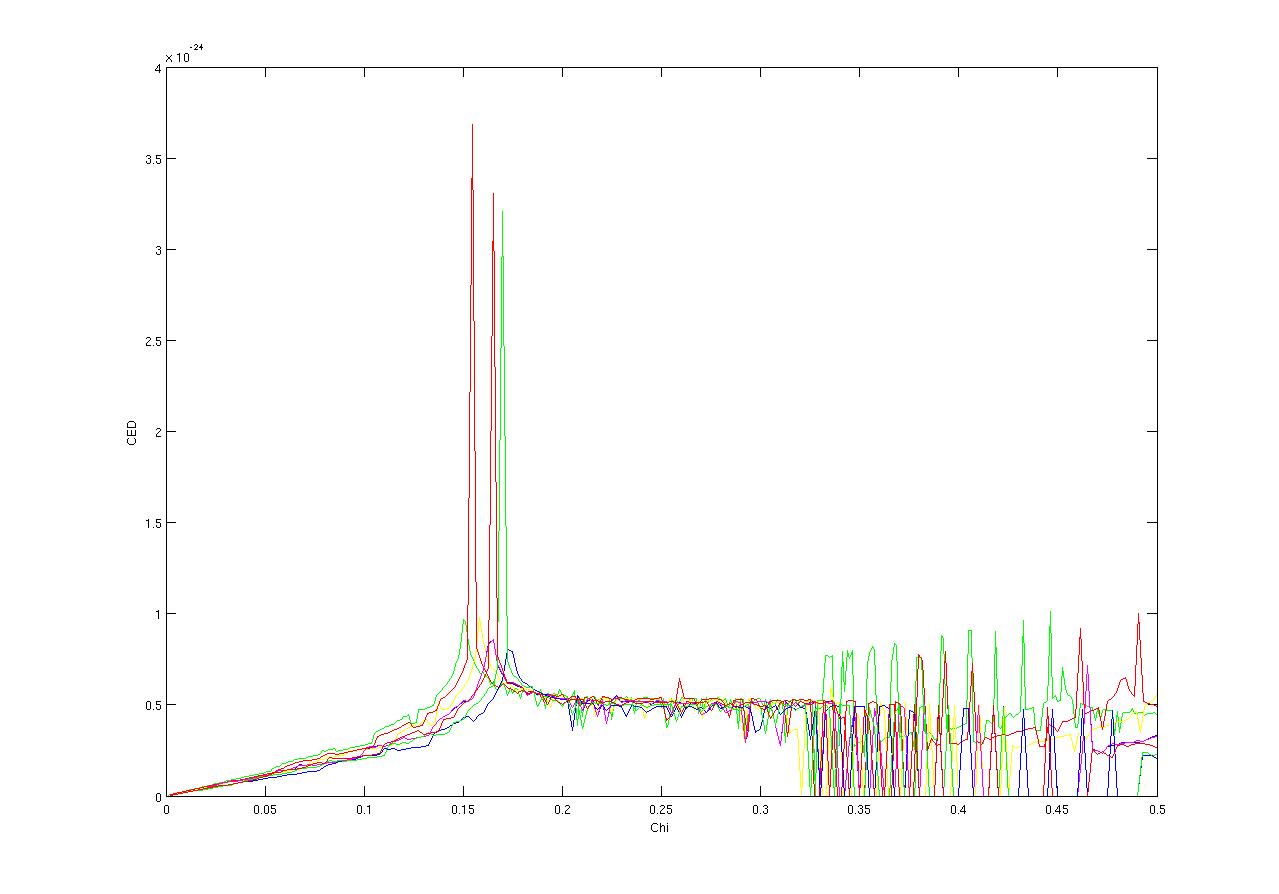
**Observations and Conclusions**

As our runs and our optimal values from our GA improved we examined the data and noticed some patterns. In our runs with the best minima, the crystal collapsed to a one-dimensional structure. This means that our and our were of equal value and our and our were also of equal value. The figure below can be examined to see that if this condition is true then the block consists of two slabs that vary in amount as is adjusted (in this one-deminsional case, does not matter). Also, if the data from the previous graph is observed one can how our and both approach the value of 14 and and approach a value of 1.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Blue** | **Red** | **Green** |
| **Min** | -0.264724 | -0.308823 | -0.313794 |
| **Ao** | 0.23272 | 0.238085 | 0.238085 |
| **Bo** | 0.227256 | 0.24172 | 0.247172 |
| **ε1** | 19.219 | 14.0159 | 14.0159 |
| **ε2** | 10.2689 | 1.18275 | 1.18277 |
| **ε3** | 1.71816 | 1.00561 | 1.00559 |
| **ε4** | 10.6683 | 13.8387 | 13.8387 |

In order to see if this one-dimensional case is indeed optimal we altered our code and added restrictions. Instead of the previous restriction in the solution to the two-dimensional cases where + must equal we also set equal to and equal to , which still satisfies this restriction but forces the GA to only test for cases of a one-dimensional structure. We ran the GA with this restriction and found minima that were as good as or often better than minima we got out of the two-dimensional crystal. Below is a chart of the minima collected using the one-dimensional algorithm and their respective graphs.

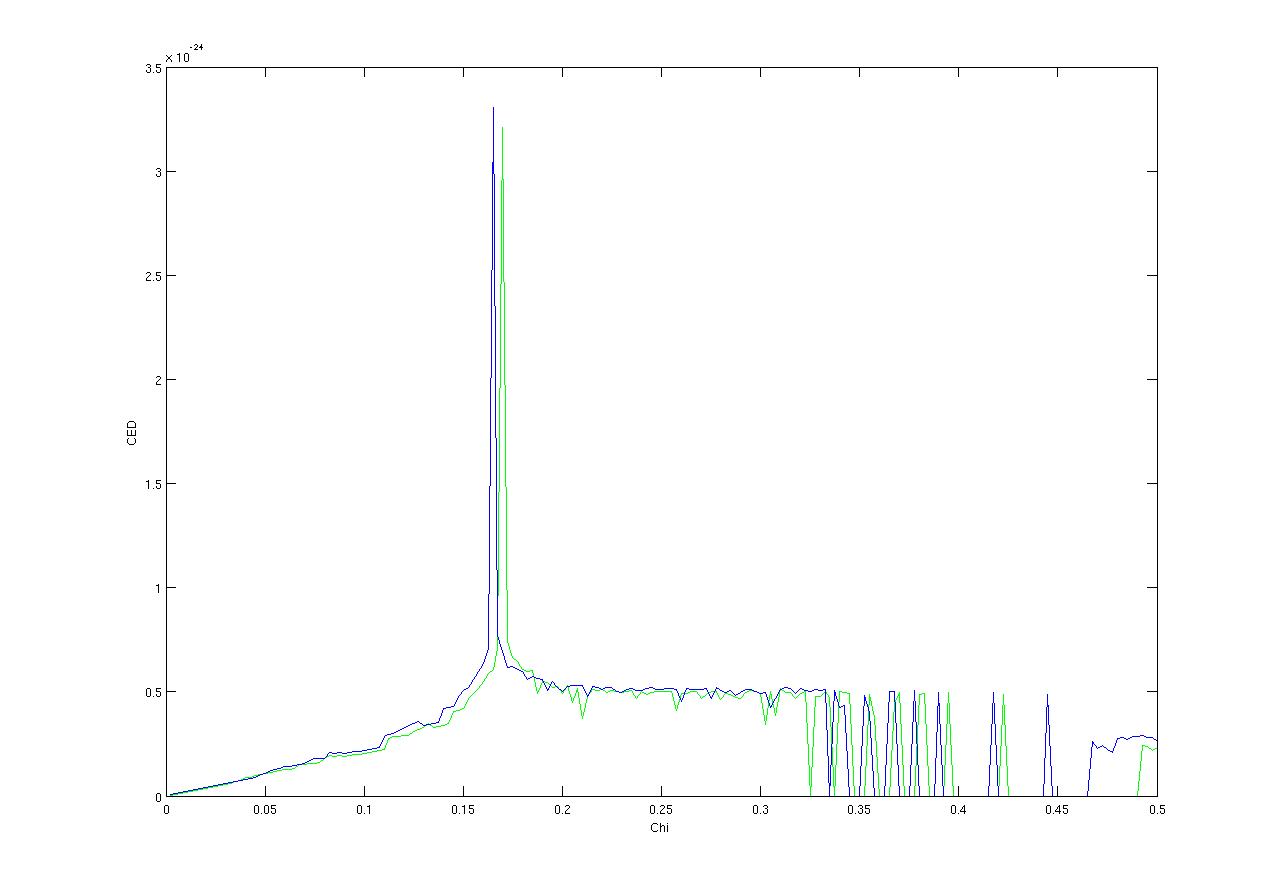
|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Run** | **Green** | **Blue** | **Yellow** | **Red** | **Magenta** | **Green Dashed** | **Red Dashed** | **Blue Dashed** |
| **min** | -0.308457 | -0.365013 | -0.356881 | -0.355223 | -0.365723 | -0.455729 | -0.443966 | -0.364322 |
| **a0** | 0.253914 | 0.212976 | 0.235392 | 0.239918 | 0.212988 | 0.193328 | 0.208801 | 0.182329 |
| **b0** | 0.010497 | 0.395225 | 0.010136 | 0.360194 | 0.487503 | 0.488508 | 0.421605 | 0.031376 |
| **ε1** | 19.534996 | 19.5931 | 19.271809 | 19.704814 | 19.590308 | 19.571342 | 19.591788 | 19.514561 |
| **ε2** | 1.985349 | 1.00732 | 1.041929 | 1.486453 | 1.007577 | 1.013816 | 1.02062 | 1.108214 |
| **ε3** | 1.985331 | 1.00732 | 1.04191 | 1.486453 | 1.0075767 | 1.010915 | 1.007353 | 1.108212 |
| **ε4** | 19.534987 | 19.5931 | 19.2718 | 19.704805 | 19.59021 | 19.568447 | 19.578521 | 19.514566 |



In addition to finding minima that were better than were found in the two-dimensional case, this one dimensional case gave us results that were very consistent. The GA repeatedly returned many high peaks near the same chi value. Also, as can be seen in the table, our epsilon values tend to contrast greatly, or as much as they can given practical limitations. In chi range of 0.1 to 0.2, our values tend towards 19 for and and 1 and . We also noticed that our best minima tend to occur when is near 0.2.

Since we were getting improved results in our one-dimensional runs, we needed to test it to make sure it was still stable in the two-dimensional algorithm. To do this we took a minimum attained using the one-dimensional algorithm and plugged it in as an initial population point in a two-dimensional algorithm. In one run (blue), we plugged in a single one-dimension minimum point as an initial population point and in another run (green), we plugged in multiple one-dimensional minima. The minima stayed consistent when run through the two-dimensional algorithm. This showed that our solutions from the one-dimensional crystals were stable. The table below respective graphs are the data from which we concluded this.

|  |  |  |
| --- | --- | --- |
| **Run** | **Blue** | **Green** |
| **Min** | -0.455728798 | -0.443966422 |
|  | 0.19332722 | 0.208801409 |
|  | 0.488508384 | 0.421605036 |
|  | 19.57134203 | 19.59178835 |
|  | 1.013815599 | 1.020620356 |
|  | 1.010915308 | 1.007353006 |
|  | 19.56844651 | 19.57852077 |

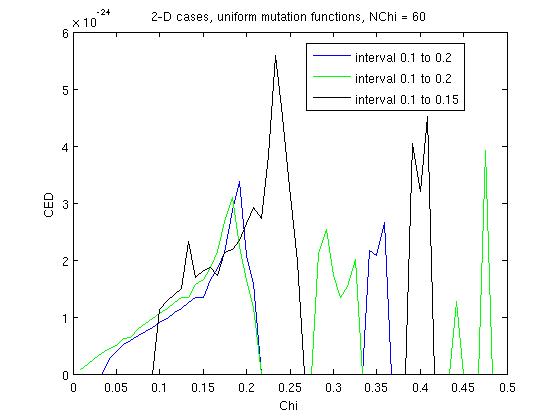


**Future Work**

One very limiting aspect of our research was the amount of time it took to run the genetic algorithm. Any reasonable solution, meaning that it was obtained with a high enough Nchi value, population size, and tolerance, took anywhere from 6 to 100 hours to obtain from the GA. As described above, we tested many different inputs of the GA to see if we could improve the result and/or efficiency, but even in the last week of our project our GA runs were taking around 20 hours to finish. Therefore, we still have variables to experiment with and test for in the future.

First, we would like to thoroughly test our conclusion about the solution to the two-dimensional crystal collapsing to a one-dimensional structure. So far this observation seems to hold true, but we don’t have enough results to compare different chi intervals, for example. We also found that using the *mutationuniform* function in Matlab’s GA returned very good optimum values. We would like to see whether the GA testing for one-dimensional solutions may also be better when using the uniform mutation function.

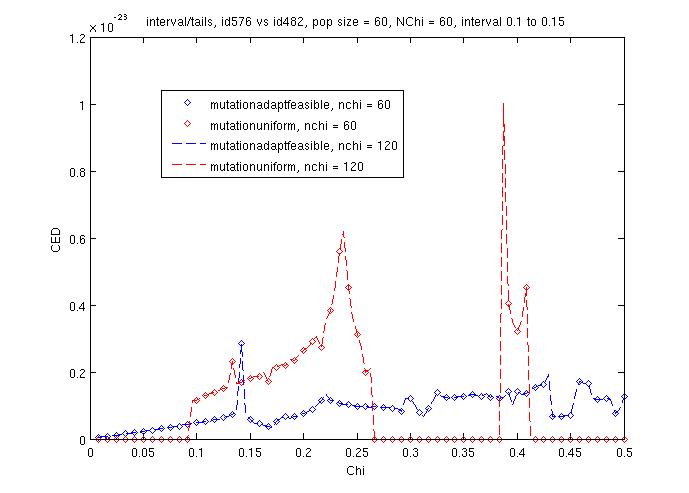
|  |  |  |  |
| --- | --- | --- | --- |
|  | **black** | **blue** | **green** |
| a0 | 0.298002 | 0.459697 | 0.31204 |
| b0 | 0.186965 | 0.302918 | 0.269292 |
| ε1 | 1.19426 | 3.58149 | 4.85406 |
| ε2 | 15.9074 | 19.3897 | 19.5415 |
| ε3 | 1.25987 | 1.85483 | 1.38539 |
| ε4 | 19.4423 | 13.796 | 16.0096 |
| min\_value | -0.173513 | -0.726828 | -0.582265 |



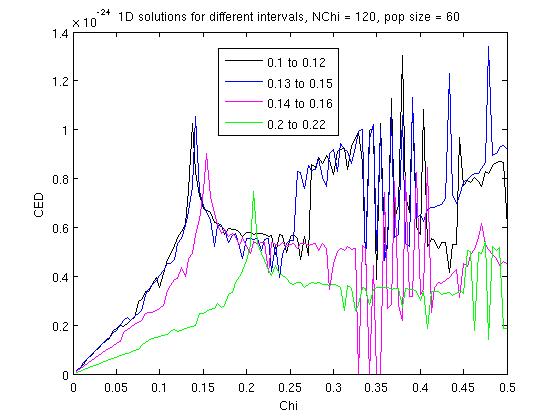
As we can see in the figure and table above, these solutions are not necessarily tending toward a 1-D

structure, but some of the minimum values are much better than those we obtained from the GA when

using the adapt feasible function.



In the figure above, we can see how one of our solutions to the 2-D structure using an adapt feasible mutation function (blue) compares to the solution using a uniform mutation function (red). Using the uniform mutation function seems to find band gaps more consistently.



In the figure to the right, we compare 1-D solutions from the GA to different chi intervals. We can see that we are indeed able to shift the chi region and still find a solution that optimizes the energy emission in that region. However, some intervals seem to produce a higher maximum CED than others.

Second, all our solutions were obtained using the restriction originally prescribed by Samokhvalova, where. To test for cases without this restriction, we would need to use a numerical software package from MIT, called the MIT Photonic Bands Package (MPB). It can find an approximate solution to the field in the anisotropic photonic crystal. Because we were not able to thoroughly test our results from the genetic algorithm, we didn’t get to move on to further testing with MPB.

Third, all our research has focused on the two-dimensional block crystal. Although we cannot say that we found one optimum solution, we have concluded that the solution collapses to a one-dimensional structure of varying dielectric material, depending on the chi region. It would be interesting to test for solutions to the three-dimensional structure, even though this adds much more complexity to the problem, as we would need to test 8 epsilon values in three-dimensions. It would be beneficial to see if there is a better solution in three-dimensions.

**Works Cited**

Greenhalgh, David, and Stephen Marshall. “Convergence Criteria for Genetic Algorithms.” Society for Industrial and Applied Mathematics 30.1 (2000): 269-282.

Kano, P., D. Baker, and M. Brio. “Analysis of the analytic dispersion relation and density of states of a selected photonic crystal.” Journal of Physics D: Applied Physics 41 (Aug. 2008).

Samokhvalova, Ksenia, Chiping Chen, and Bao-Liang Qian. “Analytical and numerical calculations of the dispersion characteristics of two-dimensional dielectric photonic band gap structures.” Journal of Applied Physics 99.063104 (2006).