Alireza Safdari Ghandhari
UIN: 728009403

Major Project
CSCE-735

12/27/2020

1. (75 points) In this assignment, you have to develop parallel code to determine the hyper-parameters $l_1$ and $l_2$ that minimize the mse. You can design an OpenMP-based shared memory code or a GPU code for this project. You are encouraged to modify the code you developed in earlier assignments for Eq. (1).

2. (20 points) Describe your strategy to parallelize the algorithm. Discuss any design choices you made to improve the parallel performance of the code. Also report on the parallel performance of your code.

These two questions are answered together in the following pages. Question 3 is answered afterward.

Over the semester Shaina Le (one of the bachelor students in our class) always recommended us to use Terra over Ada, but since I was very new to HPC, I tried not to get myself in more trouble. But after working on ada for too long and longer queue time for GPUs I decided to give Terra a shot. Since it has more cores per node (28 vs 20) and a newer GPU (k80 vs k20) with CUDA capability of 3.7 rather than 3.5 on ADA, I decided to run my codes from minor project and assignment 5 to see which one run faster, and I would pick GPU or OpenMP based on their performance.

| Task/ size | 100 X 100 | 400 X 400 | 900 X 900 | 1600 X 1600 | 2500 X 2500 | 3600 X 3600 |
|---|---|---|---|---|---|---|
| Time for LU Factorization on K80 (ms) | 1.293 | 7.487 | 45.485 | 213.46 | 768.54 | 2,243.13 |
| Time for LU Factorization and Substitution on 28 Cores (ms) | 1.94 | 15.525 | 42.237 | 48.545 | 163.46 | 1,275.99 |

As the table above shows, the 28 cores are doing the job faster and are doing more work because they are not doing just LU factorization. Moreover unlike GPU they are using double precision variables whereas GPUs are using single precision variables for all parts. I have spent a lot of time on both implementations to make them run fast and at this point I believe the task we are doing is not GPU friendly or somehow my code is not utilizing the GPU fully. One reason for this is that LU factorization has a size varying grid, and this results in divisions and modules operations to figure out the indices which take a very long time on CUDA. Also, the algorithm has a division in it which is another issue. Overall, after spending too much time trying to make my CUDA code outperform my CPU code, I am concluding that GPUs are handy when memory bus speed becomes a key factor because they can access the ram faster than CPUs. For computation, they are better only if addition and multiplication is required. I may be wrong, but I think most ridiculous claims with GPU are result of comparing the performance with a terrible single core code, or poorly optimized multi-threaded code. http://developer.download.nvidia.com/compute/cuda/1.1-Beta/x86_website/projects/reduction/doc/reduction.pdf is a study by Nvidia showing how they use hard coding and non-scaling approaches to speed things up, overall, I think a good multithreading framework on CPUs can give GPUs a hard time.

## Part 1) Grid Related Calculation

After deciding to use OpenMP, I started working on the first part of the assignment. Creating 90% training points and 10% test point. I wanted my solution to be parallel and as a result I find the number of test points per row (assuming grid of points is a matrix) and then each row is taken care of in an iteration of a parallel loop. Also, I compute grid resolution beforehand, so they are not computed for each point but for each grid position. The position is the same on both x and y axis. I was not 100% sure if parallel version of this portion of code run faster so I tested it:

| Approach\m size | m = 50 | m = 100 | m = 150 |
|---|---|---|---|
| Serial Time (us) | 40 | 165 | 321 |
| Parallel Time (us) | 8,057 | 9,804 | 7,995 |

So, as it can be seen in the table above the parallelization does not speed up the overall time, this is mainly due to the work being very small. Now that I know this part is ran in serial, I improve some of the loop.

| Approach\m size | m = 50 | m = 100 | m = 150 |
|---|---|---|---|
| Serial Time Old Approach (us) | 40 | 165 | 321 |
| Serial Time New Approach (us) | 35 | 117 | 278 |

Part 2) Computing K

In the next step I wanted to compute K for training data set. In serial version I tried two different approach, 1) using a cache unfriendly method to fill up columns as rows go forward, this is due to matrix being symmetric 2) Computing again but then writing in memory in cache friendly manner. I tested the both approach in parallel version as well.

| Approach\m size, n size | m = 50, n = 2,500 | m = 100, n = 10,000 | m = 150, n = 22,500 |
|---|---|---|---|
| Serial Time Approach 1 (ms) | 52.916 | 1,589 | 8,310 |
| Serial Time Approach 2 (ms) | 72.283 | 1,120 | 5,638 |
| Parallel Time Approach 1 (ms) | 19 | 136 | 651 |
| Parallel Time Approach 2, 1 loop parallelized (ms) | 22 | 87 | 241 |
| Parallel Time Approach 2, 2 loops parallelized (ms) | 21 | 100 | 407 |

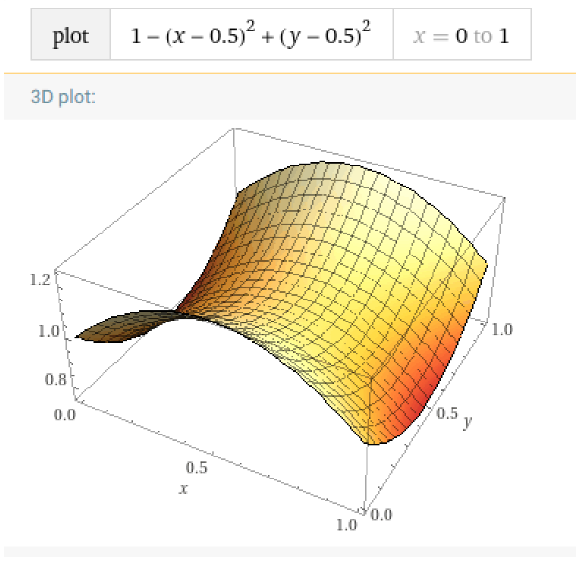
So clearly the parallel versions are better than serial versions, however we still need to pick on approach over the other one. The approach 2 is 16% slower for m = 50 while approach 1 is 56% slower for m=100 and 2701% for m=150. While approach 2 is slightly slower in lower m values, approach 1 is significantly slower for larger m values, so I stick to approach 2 with 1 loop parallelized since it showed better performance.

Part 3) Computing Observed Values

Next, I implemented observed value part in parallel, each thread computes one row. This should have been done prior to K commutation because K computation is dependent on L1 and L2 while observed value computation is not. For this assignment I use same equation as minor project.

$$f(x_i, y_i) = 1 - ((x_i - 0.5)^2 + (y_i - 0.5)^2) + d_{ij}$$

Where $d_{ij}$ is noise ranging from [-0.05, +0.05].



Source: https://www.wolframalpha.com/input/?i=1+-+%28x-0.5%29%5E2+%2B+%28y-0.5%29%5E2+from+0+to+1

Since the equation can become more complex, I just make it parallel since the benefit for very complex functions and larger m values outweigh the disadvantage for simple equation with low number of elements. Unlike the part 1 that ran better in serial, this part has a lot more calculation. For example, for m=50, part 1 had 50 commutations but this part has 50 * 50 * .90 = 2250 points that each is calculated independently. Table below shows that that the serial version is faster but due to future possibilities I keep it parallelized.

| Approach\m size | m = 50, 2,225 points | m = 100, 9,000 points | m = 150, 20,250 points |
|---|---|---|---|
| Serial Time (us) | 54 | 203 | 438 |
| Parallel Time, Second test (us) | 835 | 2,273 | 4,234 |
| Parallel Time, First test(us) | 17,293 | 11,411 | 9,157 |

For each case to consecutive runs are tested and the lower one has been selected. Huge difference was observed between the first and the second ran of parallel version, probably because in the first call, this is the first time that the code uses OpenMP and maybe there is some overhead. Also take note that the first time call reduces in each test while m increases, this could be result of my job file calling OpenMP enabled code with different values of m in ascending order. So, since m=50 is run first, it has even higher overhead compare to consecutive runs for 100 and 150. So to make the test fair, the second test result should be considered.

After this experiment I decided to repeat a small version of test in Part 2, just to see how much lower the best is performing parallel approach is:

| Approach\m size, n size | m = 50, n = 2,500 | m = 100, n = 10,000 | m = 150, n = 22,500 |
|---|---|---|---|
| Serial Time Approach 1 (ms) | 52.916 | 1,589 | 8,310 |
| Serial Time Approach 2 (ms) | 72.283 | 1,120 | 5,638 |
| Parallel Time Approach 2, 1 loop parallelized, before establishing parallel region (ms) | 22 | 87 | 241 |
| Parallel Time Approach 2, 1 loop parallelized, after establishing parallel region (ms) | 6 | 54 | 229 |

So having the observed value calculation parallel has one other benefit, it makes other parts faster.

Part 4) Adding Noise to K Matrix

For this part I just add 1 to diagonal of the K matrix. I am going to perform a Serial vs Parallel approach, since this approach is not very cache friendly and the result may be different from what we expect. But obviously it is still more efficient than adding to matrix.

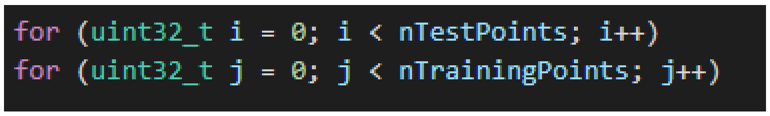
| Approach\m size | m = 50, 2,225 points | m = 100, 9,000 points | m = 150, 20,250 points |
|---|---|---|---|
| Serial Time (us) | 17 | 127 | 350 |
| Parallel Time, Second test (us) | 127 | 128 | 156 |

This result makes hard to lean either direction since it depends on m. However, assuming that LU factorization for larger m values is going to take significantly longer, it is better to optimized for lower m values in this case.

Part 5) Computation of $k_*^T$

To make sure no transpose operation is needed, I just compute the transpose version. As we saw in part 2 doing this task parallel is going to be beneficial. To do so we need two for loops, one iterating through train points and the other one iterating through test points. Obviously these two loops can have 2 possible placements. In serial there are 2, so I am going to test them, and then I test pick the winning approach for parallel version.

The for loops:



```
for (uint32_t i = 0; i < nTestPoints; i++)
for (uint32_t j = 0; j < nTrainingPoints; j++)
```

| Approach\m size | m = 50, 2250 X 250 | m = 100, 9000 X 1000 | m = 150, 20250 X 2250 |
|---|---|---|---|
| Serial Time, i in outer loop, j in inner loop (ms) | 5.524 | 86.358 | 423.023 |
| Serial Time, i in outer loop, j in inner loop (ms) | 5.109 | 88.751 | 422.415 |

Since the speed difference is not significant, I parallelized the first loop in either case. Obviously, the work in second loop is too small to be parallelized as it has been shows in part 2 of the test. So, testing only first loop in parallel is enough.

| Approach\m size | m = 50, 2250 X 250 | m = 100, 9000 X 1000 | m = 150, 20250 X 2250 |
|---|---|---|---|
| Parallel Time, i in outer loop, j in inner loop (ms) | 1.066 | 3.957 | 16.855 |
| Parallel Time, i in outer loop, j in inner loop (ms) | 1.591 | 10.553 | 57.471 |

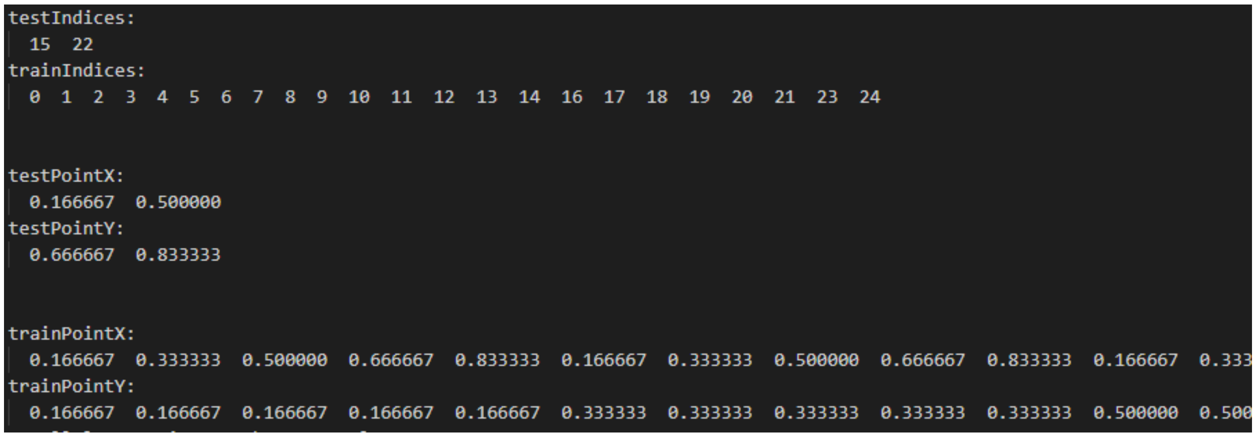
As the test suggest having i in outer loop is better. Although this is not due to cache alone, when i loop is on the outside, it has less iterations and more work per iteration. This is probably resulting in better parallelization due to lower overhead as a result lower number of threads, and lower overhead syncing after the for loop ends (there are fewer threads so the overhead for syncing is lower).

Part 6) Verification

I am going to test all the code with 5 X 5 matrix to make sure all is done correctly. I have used excel to compare results. I have checked some of the data by hand while others are compared by excel itself. I copy my matrix from output file and compare it with excel result.

| | | Point Index | | X | Y |
|---|---|---|---|---|---|
| grid | 0.166667 | 0 | 0 | 0.166667 | 0.166667 |
| m | 5 | 1 | 1 | 0.333333 | 0.166667 |
| L1 | 0.1 | 2 | 2 | 0.5 | 0.166667 |
| L2 | 0.1 | 3 | 3 | 0.666667 | 0.166667 |
| 2*L1^2 | 0.02 | 4 | 4 | 0.833333 | 0.166667 |
| 2*L2^2 | 0.02 | 5 | 0 | 0.166667 | 0.333333 |
| | | 6 | 1 | 0.333333 | 0.333333 |
| | | 7 | 2 | 0.5 | 0.333333 |
| | | 8 | 3 | 0.666667 | 0.333333 |
| | | 9 | 4 | 0.833333 | 0.333333 |
| | | 10 | 0 | 0.166667 | 0.5 |
| | | 11 | 1 | 0.333333 | 0.5 |
| | | 12 | 2 | 0.5 | 0.5 |
| | | 13 | 3 | 0.666667 | 0.5 |
| | | 14 | 4 | 0.833333 | 0.5 |
| | test | 15 | 0 | 0.166667 | 0.666667 |
| | | 16 | 1 | 0.333333 | 0.666667 |
| | | 17 | 2 | 0.5 | 0.666667 |
| | | 18 | 3 | 0.666667 | 0.666667 |
| | | 19 | 4 | 0.833333 | 0.666667 |
| | | 20 | 0 | 0.166667 | 0.833333 |
| | | 21 | 1 | 0.333333 | 0.833333 |
| | test | 22 | 2 | 0.5 | 0.833333 |
| | | 23 | 3 | 0.666667 | 0.833333 |
| | | 24 | 4 | 0.833333 | 0.833333 |

This shows the grid x and y for each point. Also the random number in Terra picked point 15 and 22 for testing. The result below is from Terra which shows the same information but computed by my code:



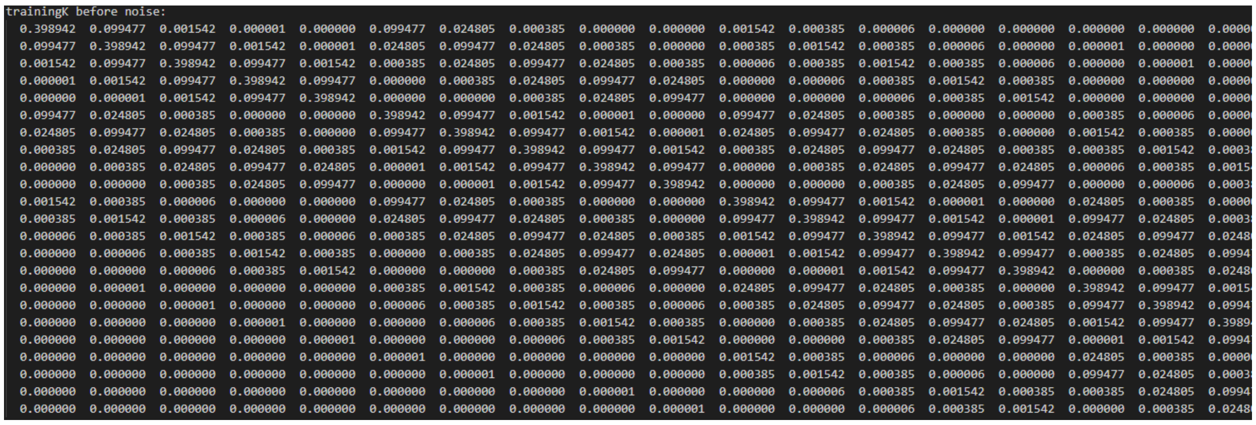
```
testIndices:
  15  22
trainIndices:
  0  1  2  3  4  5  6  7  8  9  10  11  12  13  14  16  17  18  19  20  21  23  24


testPointX:
  0.166667  0.500000
testPointY:
  0.666667  0.833333


trainPointX:
  0.166667  0.333333  0.500000  0.666667  0.833333  0.166667  0.333333  0.500000  0.666667  0.833333  0.166667  0.333
trainPointY:
  0.166667  0.166667  0.166667  0.166667  0.166667  0.333333  0.333333  0.333333  0.333333  0.333333  0.500000  0.500
```

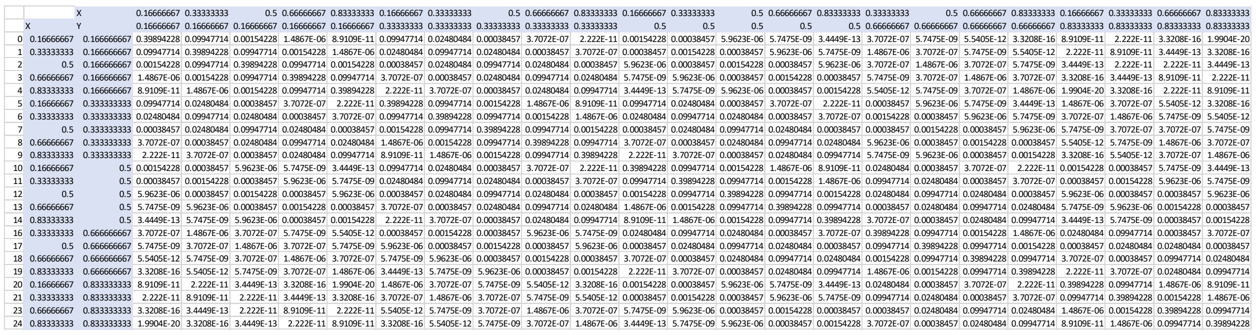
In the next part I compared my K values before adding noise. This time I have used excel to compare the values. I have created 3 tables, in the first table I compute result on excel, in second table I copy my code's result, and in the third table I compute absolute error between the first and second table for each cell. In the third table I set a threshold for error and any cell above error level become red, I show that in the following images.

K result printed by my code (partial view):



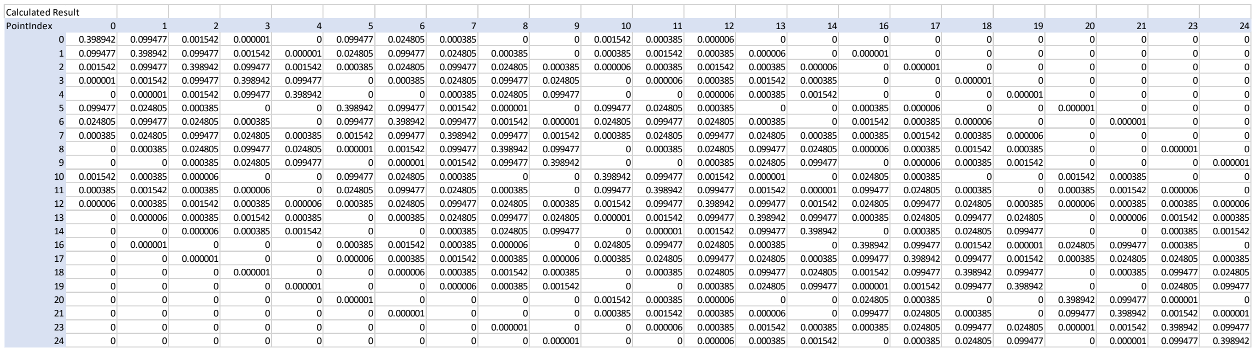
```
trainingK before noise:
  0.398942  0.099477  0.001542  0.000001  0.000000  0.099477  0.024805  0.000385  0.000000  0.000000  0.001542  0.000385  0.000006  0.000000  0.000000  0.000000  0.000000  0.0000
  0.099477  0.398942  0.099477  0.001542  0.000001  0.024805  0.099477  0.024805  0.000385  0.000000  0.000385  0.001542  0.000385  0.000006  0.000000  0.000001  0.000000  0.0000
  0.001542  0.099477  0.398942  0.099477  0.001542  0.000385  0.024805  0.099477  0.024805  0.000385  0.000006  0.000385  0.001542  0.000385  0.000006  0.000000  0.000001  0.0000
  0.000001  0.001542  0.099477  0.398942  0.099477  0.000000  0.000385  0.024805  0.099477  0.024805  0.000000  0.000006  0.000385  0.001542  0.000385  0.000000  0.000000  0.0000
  0.000000  0.000001  0.001542  0.099477  0.398942  0.000000  0.000000  0.000385  0.024805  0.099477  0.000000  0.000000  0.000006  0.000385  0.001542  0.000000  0.000000  0.0000
  0.099477  0.024805  0.000385  0.000000  0.000000  0.398942  0.099477  0.001542  0.000001  0.000000  0.099477  0.024805  0.000385  0.000000  0.000000  0.000385  0.000006  0.0000
  0.024805  0.099477  0.024805  0.000385  0.000000  0.099477  0.398942  0.099477  0.001542  0.000001  0.024805  0.099477  0.024805  0.000385  0.000000  0.001542  0.000385  0.0000
  0.000385  0.024805  0.099477  0.024805  0.000385  0.001542  0.099477  0.398942  0.099477  0.001542  0.000385  0.024805  0.099477  0.024805  0.000385  0.000385  0.001542  0.0003
  0.000000  0.000385  0.024805  0.099477  0.024805  0.000001  0.001542  0.099477  0.398942  0.099477  0.000000  0.000385  0.024805  0.099477  0.024805  0.000006  0.000385  0.0015
  0.000000  0.000000  0.000385  0.024805  0.099477  0.000000  0.000001  0.001542  0.099477  0.398942  0.000000  0.000000  0.000385  0.024805  0.099477  0.000000  0.000006  0.0003
  0.001542  0.000385  0.000006  0.000000  0.000000  0.099477  0.024805  0.000385  0.000000  0.000000  0.398942  0.099477  0.001542  0.000001  0.000000  0.024805  0.000385  0.0000
  0.000385  0.001542  0.000385  0.000006  0.000000  0.024805  0.099477  0.024805  0.000385  0.000000  0.099477  0.398942  0.099477  0.001542  0.000001  0.099477  0.024805  0.0003
  0.000006  0.000385  0.001542  0.000385  0.000006  0.000385  0.024805  0.099477  0.024805  0.000385  0.001542  0.099477  0.398942  0.099477  0.001542  0.024805  0.099477  0.0248
  0.000000  0.000006  0.000385  0.001542  0.000385  0.000000  0.000385  0.024805  0.099477  0.024805  0.000001  0.001542  0.099477  0.398942  0.099477  0.000385  0.024805  0.0994
  0.000000  0.000000  0.000006  0.000385  0.001542  0.000000  0.000000  0.000385  0.024805  0.099477  0.000000  0.000001  0.001542  0.099477  0.398942  0.000000  0.000385  0.0248
  0.000000  0.000001  0.000000  0.000000  0.000000  0.000385  0.001542  0.000385  0.000006  0.000000  0.024805  0.099477  0.024805  0.000385  0.000000  0.398942  0.099477  0.0015
  0.000000  0.000000  0.000001  0.000000  0.000000  0.000006  0.000385  0.001542  0.000385  0.000006  0.000385  0.024805  0.099477  0.024805  0.000385  0.099477  0.398942  0.0994
  0.000000  0.000000  0.000000  0.000001  0.000000  0.000000  0.000006  0.000385  0.001542  0.000385  0.000000  0.000385  0.024805  0.099477  0.024805  0.001542  0.099477  0.3989
  0.000000  0.000000  0.000000  0.000000  0.000001  0.000000  0.000000  0.000006  0.000385  0.001542  0.000000  0.000000  0.000385  0.024805  0.099477  0.000001  0.001542  0.0994
  0.000000  0.000000  0.000000  0.000000  0.000000  0.000001  0.000000  0.000000  0.000000  0.000000  0.001542  0.000385  0.000006  0.000000  0.000000  0.024805  0.000385  0.0000
  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000001  0.000000  0.000000  0.000000  0.000385  0.001542  0.000385  0.000006  0.000000  0.099477  0.024805  0.0003
  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000001  0.000000  0.000000  0.000006  0.000385  0.001542  0.000385  0.000385  0.024805  0.0994
  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000001  0.000000  0.000000  0.000006  0.000385  0.001542  0.000000  0.000385  0.0248
```

K before noise computed on Excel (I will attach the excel file for your reference):



| | | X | 0.16666667 | 0.33333333 | 0.5 | 0.66666667 | 0.83333333 | 0.16666667 | 0.33333333 | 0.5 | 0.66666667 | 0.83333333 | 0.16666667 | 0.33333333 | 0.5 | 0.66666667 | 0.83333333 | 0.33333333 | 0.5 | 0.66666667 | 0.83333333 | 0.16666667 | 0.33333333 | 0.66666667 | 0.83333333 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | X | Y | 0.16666667 | 0.16666667 | 0.16666667 | 0.16666667 | 0.16666667 | 0.33333333 | 0.33333333 | 0.33333333 | 0.33333333 | 0.33333333 | 0.5 | 0.5 | 0.5 | 0.5 | 0.5 | 0.66666667 | 0.66666667 | 0.66666667 | 0.66666667 | 0.83333333 | 0.83333333 | 0.83333333 | 0.83333333 |
| 0 | 0.16666667 | 0.166666667 | 0.39894228 | 0.09947714 | 0.00154228 | 1.4867E-06 | 8.9109E-11 | 0.09947714 | 0.02480484 | 0.00038457 | 3.7072E-07 | 2.222E-11 | 0.00154228 | 0.00038457 | 5.9623E-06 | 5.7475E-09 | 3.4449E-13 | 3.7072E-07 | 5.7475E-09 | 5.5405E-12 | 3.3208E-16 | 8.9109E-11 | 2.222E-11 | 3.3208E-16 | 1.9904E-20 |
| 1 | 0.33333333 | 0.166666667 | 0.09947714 | 0.39894228 | 0.09947714 | 0.00154228 | 1.4867E-06 | 0.02480484 | 0.09947714 | 0.02480484 | 0.00038457 | 3.7072E-07 | 0.00038457 | 0.00154228 | 0.00038457 | 5.9623E-06 | 5.7475E-09 | 1.4867E-06 | 3.7072E-07 | 5.7475E-09 | 5.5405E-12 | 2.222E-11 | 8.9109E-11 | 3.4449E-13 | 3.3208E-16 |
| 2 | 0.5 | 0.166666667 | 0.00154228 | 0.09947714 | 0.39894228 | 0.09947714 | 0.00154228 | 0.00038457 | 0.02480484 | 0.09947714 | 0.02480484 | 0.00038457 | 5.9623E-06 | 0.00038457 | 0.00154228 | 0.00038457 | 5.9623E-06 | 3.7072E-07 | 1.4867E-06 | 3.7072E-07 | 5.7475E-09 | 3.4449E-13 | 2.222E-11 | 2.222E-11 | 3.4449E-13 |
| 3 | 0.66666667 | 0.166666667 | 1.4867E-06 | 0.00154228 | 0.09947714 | 0.39894228 | 0.09947714 | 3.7072E-07 | 0.00038457 | 0.02480484 | 0.09947714 | 0.02480484 | 5.7475E-09 | 5.9623E-06 | 0.00038457 | 0.00154228 | 0.00038457 | 5.7475E-09 | 3.7072E-07 | 1.4867E-06 | 3.7072E-07 | 3.3208E-16 | 3.4449E-13 | 8.9109E-11 | 2.222E-11 |
| 4 | 0.83333333 | 0.166666667 | 8.9109E-11 | 1.4867E-06 | 0.00154228 | 0.09947714 | 0.39894228 | 2.222E-11 | 3.7072E-07 | 0.00038457 | 0.02480484 | 0.09947714 | 3.4449E-13 | 5.7475E-09 | 5.9623E-06 | 0.00038457 | 0.00154228 | 5.5405E-12 | 5.7475E-09 | 3.7072E-07 | 1.4867E-06 | 1.9904E-20 | 3.3208E-16 | 2.222E-11 | 8.9109E-11 |
| 5 | 0.16666667 | 0.333333333 | 0.09947714 | 0.02480484 | 0.00038457 | 3.7072E-07 | 2.222E-11 | 0.39894228 | 0.09947714 | 0.00154228 | 1.4867E-06 | 8.9109E-11 | 0.09947714 | 0.02480484 | 0.00038457 | 3.7072E-07 | 2.222E-11 | 0.00038457 | 5.9623E-06 | 5.7475E-09 | 3.4449E-13 | 1.4867E-06 | 3.7072E-07 | 5.5405E-12 | 3.3208E-16 |
| 6 | 0.33333333 | 0.333333333 | 0.02480484 | 0.09947714 | 0.02480484 | 0.00038457 | 3.7072E-07 | 0.09947714 | 0.39894228 | 0.09947714 | 0.00154228 | 1.4867E-06 | 0.02480484 | 0.09947714 | 0.02480484 | 0.00038457 | 3.7072E-07 | 0.00154228 | 0.00038457 | 5.9623E-06 | 5.7475E-09 | 3.7072E-07 | 1.4867E-06 | 5.7475E-09 | 5.5405E-12 |
| 7 | 0.5 | 0.333333333 | 0.00038457 | 0.02480484 | 0.09947714 | 0.02480484 | 0.00038457 | 0.00154228 | 0.09947714 | 0.39894228 | 0.09947714 | 0.00154228 | 0.00038457 | 0.02480484 | 0.09947714 | 0.02480484 | 0.00038457 | 0.00038457 | 0.00154228 | 0.00038457 | 5.9623E-06 | 5.7475E-09 | 3.7072E-07 | 3.7072E-07 | 5.7475E-09 |
| 8 | 0.66666667 | 0.333333333 | 3.7072E-07 | 0.00038457 | 0.02480484 | 0.09947714 | 0.02480484 | 1.4867E-06 | 0.00154228 | 0.09947714 | 0.39894228 | 0.09947714 | 3.7072E-07 | 0.00038457 | 0.02480484 | 0.09947714 | 0.02480484 | 5.9623E-06 | 0.00038457 | 0.00154228 | 0.00038457 | 5.5405E-12 | 5.7475E-09 | 1.4867E-06 | 3.7072E-07 |
| 9 | 0.83333333 | 0.333333333 | 2.222E-11 | 3.7072E-07 | 0.00038457 | 0.02480484 | 0.09947714 | 8.9109E-11 | 1.4867E-06 | 0.00154228 | 0.09947714 | 0.39894228 | 2.222E-11 | 3.7072E-07 | 0.00038457 | 0.02480484 | 0.09947714 | 5.7475E-09 | 5.9623E-06 | 0.00038457 | 0.00154228 | 3.3208E-16 | 5.5405E-12 | 3.7072E-07 | 1.4867E-06 |
| 10 | 0.16666667 | 0.5 | 0.00154228 | 0.00038457 | 5.9623E-06 | 5.7475E-09 | 3.4449E-13 | 0.09947714 | 0.02480484 | 0.00038457 | 3.7072E-07 | 2.222E-11 | 0.39894228 | 0.09947714 | 0.00154228 | 1.4867E-06 | 8.9109E-11 | 0.02480484 | 0.00038457 | 3.7072E-07 | 2.222E-11 | 0.00154228 | 0.00038457 | 5.7475E-09 | 3.4449E-13 |
| 11 | 0.33333333 | 0.5 | 0.00038457 | 0.00154228 | 0.00038457 | 5.9623E-06 | 5.7475E-09 | 0.02480484 | 0.09947714 | 0.02480484 | 0.00038457 | 3.7072E-07 | 0.09947714 | 0.39894228 | 0.09947714 | 0.00154228 | 1.4867E-06 | 0.09947714 | 0.02480484 | 0.00038457 | 3.7072E-07 | 0.00038457 | 0.00154228 | 5.9623E-06 | 5.7475E-09 |
| 12 | 0.5 | 0.5 | 5.9623E-06 | 0.00038457 | 0.00154228 | 0.00038457 | 5.9623E-06 | 0.00038457 | 0.02480484 | 0.09947714 | 0.02480484 | 0.00038457 | 0.00154228 | 0.09947714 | 0.39894228 | 0.09947714 | 0.00154228 | 0.02480484 | 0.09947714 | 0.02480484 | 0.00038457 | 5.9623E-06 | 0.00038457 | 0.00038457 | 5.9623E-06 |
| 13 | 0.66666667 | 0.5 | 5.7475E-09 | 5.9623E-06 | 0.00038457 | 0.00154228 | 0.00038457 | 3.7072E-07 | 0.00038457 | 0.02480484 | 0.09947714 | 0.02480484 | 1.4867E-06 | 0.00154228 | 0.09947714 | 0.39894228 | 0.09947714 | 0.00038457 | 0.02480484 | 0.09947714 | 0.02480484 | 5.7475E-09 | 5.9623E-06 | 0.00154228 | 0.00038457 |
| 14 | 0.83333333 | 0.5 | 3.4449E-13 | 5.7475E-09 | 5.9623E-06 | 0.00038457 | 0.00154228 | 2.222E-11 | 3.7072E-07 | 0.00038457 | 0.02480484 | 0.09947714 | 8.9109E-11 | 1.4867E-06 | 0.00154228 | 0.09947714 | 0.39894228 | 3.7072E-07 | 0.00038457 | 0.02480484 | 0.09947714 | 3.4449E-13 | 5.7475E-09 | 0.00038457 | 0.00154228 |
| 16 | 0.33333333 | 0.666666667 | 3.7072E-07 | 1.4867E-06 | 3.7072E-07 | 5.7475E-09 | 5.5405E-12 | 0.00038457 | 0.00154228 | 0.00038457 | 5.9623E-06 | 5.7475E-09 | 0.02480484 | 0.09947714 | 0.02480484 | 0.00038457 | 3.7072E-07 | 0.39894228 | 0.09947714 | 0.00154228 | 1.4867E-06 | 0.02480484 | 0.09947714 | 0.00038457 | 3.7072E-07 |
| 17 | 0.5 | 0.666666667 | 5.7475E-09 | 3.7072E-07 | 1.4867E-06 | 3.7072E-07 | 5.7475E-09 | 5.9623E-06 | 0.00038457 | 0.00154228 | 0.00038457 | 5.9623E-06 | 0.00038457 | 0.02480484 | 0.09947714 | 0.02480484 | 0.00038457 | 0.09947714 | 0.39894228 | 0.09947714 | 0.00154228 | 0.00038457 | 0.02480484 | 0.02480484 | 0.00038457 |
| 18 | 0.66666667 | 0.666666667 | 5.5405E-12 | 5.7475E-09 | 3.7072E-07 | 1.4867E-06 | 3.7072E-07 | 5.7475E-09 | 5.9623E-06 | 0.00038457 | 0.00154228 | 0.00038457 | 3.7072E-07 | 0.00038457 | 0.02480484 | 0.09947714 | 0.02480484 | 0.00154228 | 0.09947714 | 0.39894228 | 0.09947714 | 3.7072E-07 | 0.00038457 | 0.09947714 | 0.02480484 |
| 19 | 0.83333333 | 0.666666667 | 3.3208E-16 | 5.5405E-12 | 5.7475E-09 | 3.7072E-07 | 1.4867E-06 | 3.4449E-13 | 5.7475E-09 | 5.9623E-06 | 0.00038457 | 0.00154228 | 2.222E-11 | 3.7072E-07 | 0.00038457 | 0.02480484 | 0.09947714 | 1.4867E-06 | 0.00154228 | 0.09947714 | 0.39894228 | 2.222E-11 | 3.7072E-07 | 0.02480484 | 0.09947714 |
| 20 | 0.16666667 | 0.833333333 | 8.9109E-11 | 2.222E-11 | 3.4449E-13 | 3.3208E-16 | 1.9904E-20 | 1.4867E-06 | 3.7072E-07 | 5.7475E-09 | 5.5405E-12 | 3.3208E-16 | 0.00154228 | 0.00038457 | 5.9623E-06 | 5.7475E-09 | 3.4449E-13 | 0.02480484 | 0.00038457 | 3.7072E-07 | 2.222E-11 | 0.39894228 | 0.09947714 | 1.4867E-06 | 8.9109E-11 |
| 21 | 0.33333333 | 0.833333333 | 2.222E-11 | 8.9109E-11 | 2.222E-11 | 3.4449E-13 | 3.3208E-16 | 3.7072E-07 | 1.4867E-06 | 3.7072E-07 | 5.7475E-09 | 5.5405E-12 | 0.00038457 | 0.00154228 | 0.00038457 | 5.9623E-06 | 5.7475E-09 | 0.09947714 | 0.02480484 | 0.00038457 | 3.7072E-07 | 0.09947714 | 0.39894228 | 0.00154228 | 1.4867E-06 |
| 23 | 0.66666667 | 0.833333333 | 3.3208E-16 | 3.4449E-13 | 2.222E-11 | 8.9109E-11 | 2.222E-11 | 5.5405E-12 | 5.7475E-09 | 3.7072E-07 | 1.4867E-06 | 3.7072E-07 | 5.7475E-09 | 5.9623E-06 | 0.00038457 | 0.00154228 | 0.00038457 | 0.00038457 | 0.02480484 | 0.09947714 | 0.02480484 | 1.4867E-06 | 0.00154228 | 0.39894228 | 0.09947714 |
| 24 | 0.83333333 | 0.833333333 | 1.9904E-20 | 3.3208E-16 | 3.4449E-13 | 2.222E-11 | 8.9109E-11 | 3.3208E-16 | 5.5405E-12 | 5.7475E-09 | 3.7072E-07 | 1.4867E-06 | 3.4449E-13 | 5.7475E-09 | 5.9623E-06 | 0.00038457 | 0.00154228 | 3.7072E-07 | 0.00038457 | 0.02480484 | 0.09947714 | 8.9109E-11 | 1.4867E-06 | 0.09947714 | 0.39894228 |

K before noise computed by my code:



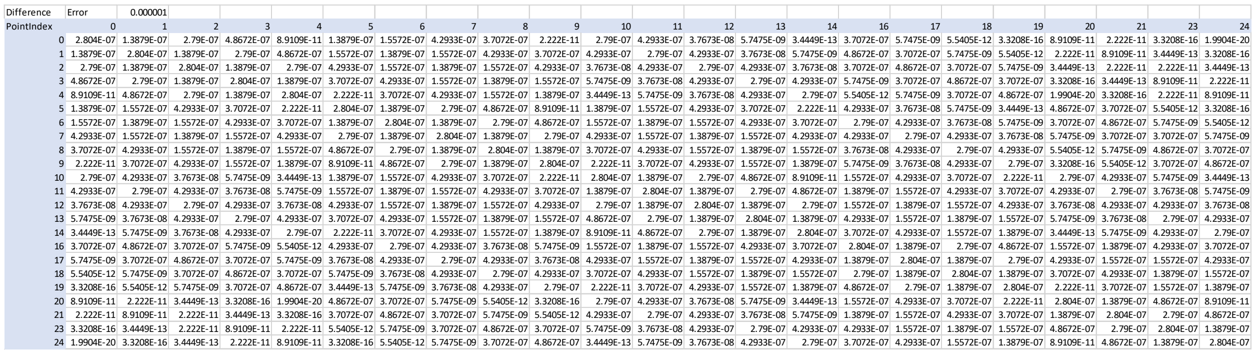
| Calculated Result | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| PointIndex | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 16 | 17 | 18 | 19 | 20 | 21 | 23 | 24 |
| 0 | 0.398942 | 0.099477 | 0.001542 | 0.000001 | 0 | 0.099477 | 0.024805 | 0.000385 | 0 | 0 | 0.001542 | 0.000385 | 0.000006 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0.099477 | 0.398942 | 0.099477 | 0.001542 | 0.000001 | 0.024805 | 0.099477 | 0.024805 | 0.000385 | 0 | 0.000385 | 0.001542 | 0.000385 | 0.000006 | 0 | 0.000001 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 0.001542 | 0.099477 | 0.398942 | 0.099477 | 0.001542 | 0.000385 | 0.024805 | 0.099477 | 0.024805 | 0.000385 | 0.000006 | 0.000385 | 0.001542 | 0.000385 | 0.000006 | 0 | 0.000001 | 0 | 0 | 0 | 0 | 0 | 0 |
| 3 | 0.000001 | 0.001542 | 0.099477 | 0.398942 | 0.099477 | 0 | 0.000385 | 0.024805 | 0.099477 | 0.024805 | 0 | 0.000006 | 0.000385 | 0.001542 | 0.000385 | 0 | 0 | 0.000001 | 0 | 0 | 0 | 0 | 0 |
| 4 | 0 | 0.000001 | 0.001542 | 0.099477 | 0.398942 | 0 | 0 | 0.000385 | 0.024805 | 0.099477 | 0 | 0 | 0.000006 | 0.000385 | 0.001542 | 0 | 0 | 0 | 0.000001 | 0 | 0 | 0 | 0 |
| 5 | 0.099477 | 0.024805 | 0.000385 | 0 | 0 | 0.398942 | 0.099477 | 0.001542 | 0.000001 | 0 | 0.099477 | 0.024805 | 0.000385 | 0 | 0 | 0.000385 | 0.000006 | 0 | 0 | 0.000001 | 0 | 0 | 0 |
| 6 | 0.024805 | 0.099477 | 0.024805 | 0.000385 | 0 | 0.099477 | 0.398942 | 0.099477 | 0.001542 | 0.000001 | 0.024805 | 0.099477 | 0.024805 | 0.000385 | 0 | 0.001542 | 0.000385 | 0.000006 | 0 | 0 | 0.000001 | 0 | 0 |
| 7 | 0.000385 | 0.024805 | 0.099477 | 0.024805 | 0.000385 | 0.001542 | 0.099477 | 0.398942 | 0.099477 | 0.001542 | 0.000385 | 0.024805 | 0.099477 | 0.024805 | 0.000385 | 0.000385 | 0.001542 | 0.000385 | 0.000006 | 0 | 0 | 0 | 0 |
| 8 | 0 | 0.000385 | 0.024805 | 0.099477 | 0.024805 | 0.000001 | 0.001542 | 0.099477 | 0.398942 | 0.099477 | 0 | 0.000385 | 0.024805 | 0.099477 | 0.024805 | 0.000006 | 0.000385 | 0.001542 | 0.000385 | 0 | 0 | 0.000001 | 0 |
| 9 | 0 | 0 | 0.000385 | 0.024805 | 0.099477 | 0 | 0.000001 | 0.001542 | 0.099477 | 0.398942 | 0 | 0 | 0.000385 | 0.024805 | 0.099477 | 0 | 0.000006 | 0.000385 | 0.001542 | 0 | 0 | 0 | 0.000001 |
| 10 | 0.001542 | 0.000385 | 0.000006 | 0 | 0 | 0.099477 | 0.024805 | 0.000385 | 0 | 0 | 0.398942 | 0.099477 | 0.001542 | 0.000001 | 0 | 0.024805 | 0.000385 | 0 | 0 | 0.001542 | 0.000385 | 0 | 0 |
| 11 | 0.000385 | 0.001542 | 0.000385 | 0.000006 | 0 | 0.024805 | 0.099477 | 0.024805 | 0.000385 | 0 | 0.099477 | 0.398942 | 0.099477 | 0.001542 | 0.000001 | 0.099477 | 0.024805 | 0.000385 | 0 | 0.000385 | 0.001542 | 0.000006 | 0 |
| 12 | 0.000006 | 0.000385 | 0.001542 | 0.000385 | 0.000006 | 0.000385 | 0.024805 | 0.099477 | 0.024805 | 0.000385 | 0.001542 | 0.099477 | 0.398942 | 0.099477 | 0.001542 | 0.024805 | 0.099477 | 0.024805 | 0.000385 | 0.000006 | 0.000385 | 0.000385 | 0.000006 |
| 13 | 0 | 0.000006 | 0.000385 | 0.001542 | 0.000385 | 0 | 0.000385 | 0.024805 | 0.099477 | 0.024805 | 0.000001 | 0.001542 | 0.099477 | 0.398942 | 0.099477 | 0.000385 | 0.024805 | 0.099477 | 0.024805 | 0 | 0.000006 | 0.001542 | 0.000385 |
| 14 | 0 | 0 | 0.000006 | 0.000385 | 0.001542 | 0 | 0 | 0.000385 | 0.024805 | 0.099477 | 0 | 0.000001 | 0.001542 | 0.099477 | 0.398942 | 0 | 0.000385 | 0.024805 | 0.099477 | 0 | 0 | 0.000385 | 0.001542 |
| 16 | 0 | 0.000001 | 0 | 0 | 0 | 0.000385 | 0.001542 | 0.000385 | 0.000006 | 0 | 0.024805 | 0.099477 | 0.024805 | 0.000385 | 0 | 0.398942 | 0.099477 | 0.001542 | 0.000001 | 0.024805 | 0.099477 | 0.000385 | 0 |
| 17 | 0 | 0 | 0.000001 | 0 | 0 | 0.000006 | 0.000385 | 0.001542 | 0.000385 | 0.000006 | 0.000385 | 0.024805 | 0.099477 | 0.024805 | 0.000385 | 0.099477 | 0.398942 | 0.099477 | 0.001542 | 0.000385 | 0.024805 | 0.024805 | 0.000385 |
| 18 | 0 | 0 | 0 | 0.000001 | 0 | 0 | 0.000006 | 0.000385 | 0.001542 | 0.000385 | 0 | 0.000385 | 0.024805 | 0.099477 | 0.024805 | 0.001542 | 0.099477 | 0.398942 | 0.099477 | 0 | 0.000385 | 0.099477 | 0.024805 |
| 19 | 0 | 0 | 0 | 0 | 0.000001 | 0 | 0 | 0.000006 | 0.000385 | 0.001542 | 0 | 0 | 0.000385 | 0.024805 | 0.099477 | 0.000001 | 0.001542 | 0.099477 | 0.398942 | 0 | 0 | 0.024805 | 0.099477 |
| 20 | 0 | 0 | 0 | 0 | 0 | 0.000001 | 0 | 0 | 0 | 0 | 0.001542 | 0.000385 | 0.000006 | 0 | 0 | 0.024805 | 0.000385 | 0 | 0 | 0.398942 | 0.099477 | 0.000001 | 0 |
| 21 | 0 | 0 | 0 | 0 | 0 | 0 | 0.000001 | 0 | 0 | 0 | 0.000385 | 0.001542 | 0.000385 | 0.000006 | 0 | 0.099477 | 0.024805 | 0.000385 | 0 | 0.099477 | 0.398942 | 0.001542 | 0.000001 |
| 23 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0.000001 | 0 | 0 | 0.000006 | 0.000385 | 0.001542 | 0.000385 | 0.000385 | 0.024805 | 0.099477 | 0.024805 | 0.000001 | 0.001542 | 0.398942 | 0.099477 |
| 24 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0.000001 | 0 | 0 | 0.000006 | 0.000385 | 0.001542 | 0 | 0.000385 | 0.024805 | 0.099477 | 0 | 0.000001 | 0.099477 | 0.398942 |

Alireza Safdari Ghandhari
UIN: 728009403

Major Project
CSCE-735

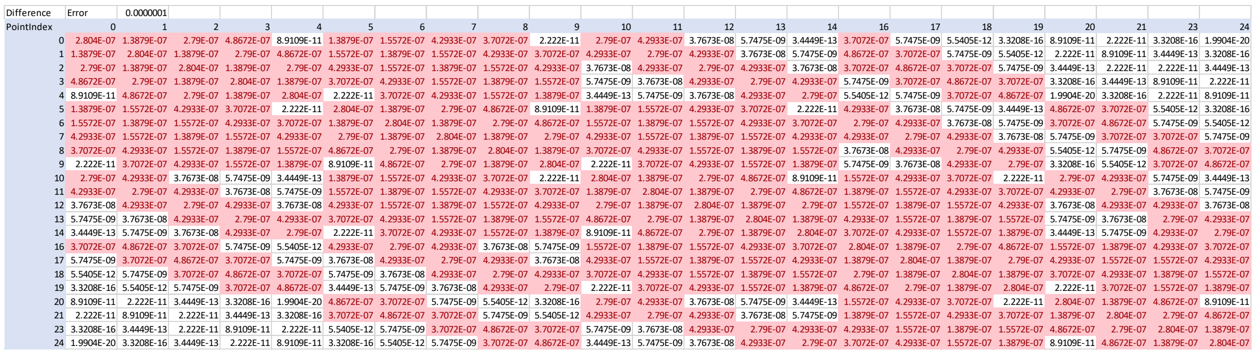
12/27/2020

K before noise, absolute error between my code and excel calculation (error above 0.000001 is turned red): No error above 0.000001 since there are no red cells



| Difference | Error | 0.000001 | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| PointIndex | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 16 | 17 | 18 | 19 | 20 | 21 | 23 | 24 |
| 0 | 2.804E-07 | 1.3879E-07 | 2.79E-07 | 4.8672E-07 | 8.9109E-11 | 1.3879E-07 | 1.5572E-07 | 4.2933E-07 | 3.7072E-07 | 2.222E-11 | 2.79E-07 | 4.2933E-07 | 3.7673E-08 | 5.7475E-09 | 3.4449E-13 | 3.7072E-07 | 5.7475E-09 | 5.5405E-12 | 3.3208E-16 | 8.9109E-11 | 2.222E-11 | 3.3208E-16 | 1.9904E-20 |
| 1 | 1.3879E-07 | 2.804E-07 | 1.3879E-07 | 2.79E-07 | 4.8672E-07 | 1.5572E-07 | 1.3879E-07 | 1.5572E-07 | 4.2933E-07 | 3.7072E-07 | 4.2933E-07 | 2.79E-07 | 4.2933E-07 | 3.7673E-08 | 5.7475E-09 | 4.8672E-07 | 3.7072E-07 | 5.7475E-09 | 5.5405E-12 | 2.222E-11 | 8.9109E-11 | 3.4449E-13 | 3.3208E-16 |
| 2 | 2.79E-07 | 1.3879E-07 | 2.804E-07 | 1.3879E-07 | 2.79E-07 | 4.2933E-07 | 1.5572E-07 | 1.3879E-07 | 1.5572E-07 | 4.2933E-07 | 3.7673E-08 | 4.2933E-07 | 2.79E-07 | 4.2933E-07 | 3.7673E-08 | 3.7072E-07 | 4.8672E-07 | 3.7072E-07 | 5.7475E-09 | 3.4449E-13 | 2.222E-11 | 2.222E-11 | 3.4449E-13 |
| 3 | 4.8672E-07 | 2.79E-07 | 1.3879E-07 | 2.804E-07 | 1.3879E-07 | 3.7072E-07 | 4.2933E-07 | 1.5572E-07 | 1.3879E-07 | 1.5572E-07 | 5.7475E-09 | 3.7673E-08 | 4.2933E-07 | 2.79E-07 | 4.2933E-07 | 5.7475E-09 | 3.7072E-07 | 4.8672E-07 | 3.7072E-07 | 3.3208E-16 | 3.4449E-13 | 8.9109E-11 | 2.222E-11 |
| 4 | 8.9109E-11 | 4.8672E-07 | 2.79E-07 | 1.3879E-07 | 2.804E-07 | 2.222E-11 | 3.7072E-07 | 4.2933E-07 | 1.5572E-07 | 1.3879E-07 | 3.4449E-13 | 5.7475E-09 | 3.7673E-08 | 4.2933E-07 | 2.79E-07 | 5.5405E-12 | 5.7475E-09 | 3.7072E-07 | 4.8672E-07 | 1.9904E-20 | 3.3208E-16 | 2.222E-11 | 8.9109E-11 |
| 5 | 1.3879E-07 | 1.5572E-07 | 4.2933E-07 | 3.7072E-07 | 2.222E-11 | 2.804E-07 | 1.3879E-07 | 2.79E-07 | 4.8672E-07 | 8.9109E-11 | 1.3879E-07 | 1.5572E-07 | 4.2933E-07 | 3.7072E-07 | 2.222E-11 | 4.2933E-07 | 3.7673E-08 | 5.7475E-09 | 3.4449E-13 | 4.8672E-07 | 3.7072E-07 | 5.5405E-12 | 3.3208E-16 |
| 6 | 1.5572E-07 | 1.3879E-07 | 1.5572E-07 | 4.2933E-07 | 3.7072E-07 | 1.3879E-07 | 2.804E-07 | 1.3879E-07 | 2.79E-07 | 4.8672E-07 | 1.5572E-07 | 1.3879E-07 | 1.5572E-07 | 4.2933E-07 | 3.7072E-07 | 2.79E-07 | 4.2933E-07 | 3.7673E-08 | 5.7475E-09 | 3.7072E-07 | 4.8672E-07 | 5.7475E-09 | 5.5405E-12 |
| 7 | 4.2933E-07 | 1.5572E-07 | 1.3879E-07 | 1.5572E-07 | 4.2933E-07 | 2.79E-07 | 1.3879E-07 | 2.804E-07 | 1.3879E-07 | 2.79E-07 | 4.2933E-07 | 1.5572E-07 | 1.3879E-07 | 1.5572E-07 | 4.2933E-07 | 4.2933E-07 | 2.79E-07 | 4.2933E-07 | 3.7673E-08 | 5.7475E-09 | 3.7072E-07 | 3.7072E-07 | 5.7475E-09 |
| 8 | 3.7072E-07 | 4.2933E-07 | 1.5572E-07 | 1.3879E-07 | 1.5572E-07 | 4.8672E-07 | 2.79E-07 | 1.3879E-07 | 2.804E-07 | 1.3879E-07 | 3.7072E-07 | 4.2933E-07 | 1.5572E-07 | 1.3879E-07 | 1.5572E-07 | 3.7673E-08 | 4.2933E-07 | 2.79E-07 | 4.2933E-07 | 5.5405E-12 | 5.7475E-09 | 4.8672E-07 | 3.7072E-07 |
| 9 | 2.222E-11 | 3.7072E-07 | 4.2933E-07 | 1.5572E-07 | 1.3879E-07 | 8.9109E-11 | 4.8672E-07 | 2.79E-07 | 1.3879E-07 | 2.804E-07 | 2.222E-11 | 3.7072E-07 | 4.2933E-07 | 1.5572E-07 | 1.3879E-07 | 5.7475E-09 | 3.7673E-08 | 4.2933E-07 | 2.79E-07 | 3.3208E-16 | 5.5405E-12 | 3.7072E-07 | 4.8672E-07 |
| 10 | 2.79E-07 | 4.2933E-07 | 3.7673E-08 | 5.7475E-09 | 3.4449E-13 | 1.3879E-07 | 1.5572E-07 | 4.2933E-07 | 3.7072E-07 | 2.222E-11 | 2.804E-07 | 1.3879E-07 | 2.79E-07 | 4.8672E-07 | 8.9109E-11 | 1.5572E-07 | 4.2933E-07 | 3.7072E-07 | 2.222E-11 | 2.79E-07 | 4.2933E-07 | 5.7475E-09 | 3.4449E-13 |
| 11 | 4.2933E-07 | 2.79E-07 | 4.2933E-07 | 3.7673E-08 | 5.7475E-09 | 1.5572E-07 | 1.3879E-07 | 1.5572E-07 | 4.2933E-07 | 3.7072E-07 | 1.3879E-07 | 2.804E-07 | 1.3879E-07 | 2.79E-07 | 4.8672E-07 | 1.3879E-07 | 1.5572E-07 | 4.2933E-07 | 3.7072E-07 | 4.2933E-07 | 2.79E-07 | 3.7673E-08 | 5.7475E-09 |
| 12 | 3.7673E-08 | 4.2933E-07 | 2.79E-07 | 4.2933E-07 | 3.7673E-08 | 4.2933E-07 | 1.5572E-07 | 1.3879E-07 | 1.5572E-07 | 4.2933E-07 | 2.79E-07 | 1.3879E-07 | 2.804E-07 | 1.3879E-07 | 2.79E-07 | 1.5572E-07 | 1.3879E-07 | 1.5572E-07 | 4.2933E-07 | 3.7673E-08 | 4.2933E-07 | 4.2933E-07 | 3.7673E-08 |
| 13 | 5.7475E-09 | 3.7673E-08 | 4.2933E-07 | 2.79E-07 | 4.2933E-07 | 3.7072E-07 | 4.2933E-07 | 1.5572E-07 | 1.3879E-07 | 1.5572E-07 | 4.8672E-07 | 2.79E-07 | 1.3879E-07 | 2.804E-07 | 1.3879E-07 | 4.2933E-07 | 1.5572E-07 | 1.3879E-07 | 1.5572E-07 | 5.7475E-09 | 3.7673E-08 | 2.79E-07 | 4.2933E-07 |
| 14 | 3.4449E-13 | 5.7475E-09 | 3.7673E-08 | 4.2933E-07 | 2.79E-07 | 2.222E-11 | 3.7072E-07 | 4.2933E-07 | 1.5572E-07 | 1.3879E-07 | 8.9109E-11 | 4.8672E-07 | 2.79E-07 | 1.3879E-07 | 2.804E-07 | 3.7072E-07 | 4.2933E-07 | 1.5572E-07 | 1.3879E-07 | 3.4449E-13 | 5.7475E-09 | 4.2933E-07 | 2.79E-07 |
| 16 | 3.7072E-07 | 4.8672E-07 | 3.7072E-07 | 5.7475E-09 | 5.5405E-12 | 4.2933E-07 | 2.79E-07 | 4.2933E-07 | 3.7673E-08 | 5.7475E-09 | 1.5572E-07 | 1.3879E-07 | 1.5572E-07 | 4.2933E-07 | 3.7072E-07 | 2.804E-07 | 1.3879E-07 | 2.79E-07 | 4.8672E-07 | 1.5572E-07 | 1.3879E-07 | 4.2933E-07 | 3.7072E-07 |
| 17 | 5.7475E-09 | 3.7072E-07 | 4.8672E-07 | 3.7072E-07 | 5.7475E-09 | 3.7673E-08 | 4.2933E-07 | 2.79E-07 | 4.2933E-07 | 3.7673E-08 | 4.2933E-07 | 1.5572E-07 | 1.3879E-07 | 1.5572E-07 | 4.2933E-07 | 1.3879E-07 | 2.804E-07 | 1.3879E-07 | 2.79E-07 | 4.2933E-07 | 1.5572E-07 | 1.5572E-07 | 4.2933E-07 |
| 18 | 5.5405E-12 | 5.7475E-09 | 3.7072E-07 | 4.8672E-07 | 3.7072E-07 | 5.7475E-09 | 3.7673E-08 | 4.2933E-07 | 2.79E-07 | 4.2933E-07 | 3.7072E-07 | 4.2933E-07 | 1.5572E-07 | 1.3879E-07 | 1.5572E-07 | 2.79E-07 | 1.3879E-07 | 2.804E-07 | 1.3879E-07 | 3.7072E-07 | 4.2933E-07 | 1.3879E-07 | 1.5572E-07 |
| 19 | 3.3208E-16 | 5.5405E-12 | 5.7475E-09 | 3.7072E-07 | 4.8672E-07 | 3.4449E-13 | 5.7475E-09 | 3.7673E-08 | 4.2933E-07 | 2.79E-07 | 2.222E-11 | 3.7072E-07 | 4.2933E-07 | 1.5572E-07 | 1.3879E-07 | 4.8672E-07 | 2.79E-07 | 1.3879E-07 | 2.804E-07 | 2.222E-11 | 3.7072E-07 | 1.5572E-07 | 1.3879E-07 |
| 20 | 8.9109E-11 | 2.222E-11 | 3.4449E-13 | 3.3208E-16 | 1.9904E-20 | 4.8672E-07 | 3.7072E-07 | 5.7475E-09 | 5.5405E-12 | 3.3208E-16 | 2.79E-07 | 4.2933E-07 | 3.7673E-08 | 5.7475E-09 | 3.4449E-13 | 1.5572E-07 | 4.2933E-07 | 3.7072E-07 | 2.222E-11 | 2.804E-07 | 1.3879E-07 | 4.8672E-07 | 8.9109E-11 |
| 21 | 2.222E-11 | 8.9109E-11 | 2.222E-11 | 3.4449E-13 | 3.3208E-16 | 3.7072E-07 | 4.8672E-07 | 3.7072E-07 | 5.7475E-09 | 5.5405E-12 | 4.2933E-07 | 2.79E-07 | 4.2933E-07 | 3.7673E-08 | 5.7475E-09 | 1.3879E-07 | 1.5572E-07 | 4.2933E-07 | 3.7072E-07 | 1.3879E-07 | 2.804E-07 | 2.79E-07 | 4.8672E-07 |
| 23 | 3.3208E-16 | 3.4449E-13 | 2.222E-11 | 8.9109E-11 | 2.222E-11 | 5.5405E-12 | 5.7475E-09 | 3.7072E-07 | 4.8672E-07 | 3.7072E-07 | 5.7475E-09 | 3.7673E-08 | 4.2933E-07 | 2.79E-07 | 4.2933E-07 | 4.2933E-07 | 1.5572E-07 | 1.3879E-07 | 1.5572E-07 | 4.8672E-07 | 2.79E-07 | 2.804E-07 | 1.3879E-07 |
| 24 | 1.9904E-20 | 3.3208E-16 | 3.4449E-13 | 2.222E-11 | 8.9109E-11 | 3.3208E-16 | 5.5405E-12 | 5.7475E-09 | 3.7072E-07 | 4.8672E-07 | 3.4449E-13 | 5.7475E-09 | 3.7673E-08 | 4.2933E-07 | 2.79E-07 | 3.7072E-07 | 4.2933E-07 | 1.5572E-07 | 1.3879E-07 | 8.9109E-11 | 4.8672E-07 | 1.3879E-07 | 2.804E-07 |

K before noise, absolute error between my code and excel calculation (error above 0.0000001 is turned red): many cells turned red showing error above 0.0000001

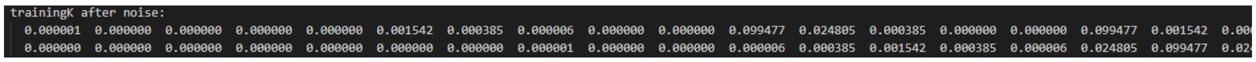


| Difference | Error | 0.0000001 | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| PointIndex | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 16 | 17 | 18 | 19 | 20 | 21 | 23 | 24 |
| 0 | 2.804E-07 | 1.3879E-07 | 2.79E-07 | 4.8672E-07 | 8.9109E-11 | 1.3879E-07 | 1.5572E-07 | 4.2933E-07 | 3.7072E-07 | 2.222E-11 | 2.79E-07 | 4.2933E-07 | 3.7673E-08 | 5.7475E-09 | 3.4449E-13 | 3.7072E-07 | 5.7475E-09 | 5.5405E-12 | 3.3208E-16 | 8.9109E-11 | 2.222E-11 | 3.3208E-16 | 1.9904E-20 |
| 1 | 1.3879E-07 | 2.804E-07 | 1.3879E-07 | 2.79E-07 | 4.8672E-07 | 1.5572E-07 | 1.3879E-07 | 1.5572E-07 | 4.2933E-07 | 3.7072E-07 | 4.2933E-07 | 2.79E-07 | 4.2933E-07 | 3.7673E-08 | 5.7475E-09 | 4.8672E-07 | 3.7072E-07 | 5.7475E-09 | 5.5405E-12 | 2.222E-11 | 8.9109E-11 | 3.4449E-13 | 3.3208E-16 |
| 2 | 2.79E-07 | 1.3879E-07 | 2.804E-07 | 1.3879E-07 | 2.79E-07 | 4.2933E-07 | 1.5572E-07 | 1.3879E-07 | 1.5572E-07 | 4.2933E-07 | 3.7673E-08 | 4.2933E-07 | 2.79E-07 | 4.2933E-07 | 3.7673E-08 | 3.7072E-07 | 4.8672E-07 | 3.7072E-07 | 5.7475E-09 | 3.4449E-13 | 2.222E-11 | 2.222E-11 | 3.4449E-13 |
| 3 | 4.8672E-07 | 2.79E-07 | 1.3879E-07 | 2.804E-07 | 1.3879E-07 | 3.7072E-07 | 4.2933E-07 | 1.5572E-07 | 1.3879E-07 | 1.5572E-07 | 5.7475E-09 | 3.7673E-08 | 4.2933E-07 | 2.79E-07 | 4.2933E-07 | 5.7475E-09 | 3.7072E-07 | 4.8672E-07 | 3.7072E-07 | 3.3208E-16 | 3.4449E-13 | 8.9109E-11 | 2.222E-11 |
| 4 | 8.9109E-11 | 4.8672E-07 | 2.79E-07 | 1.3879E-07 | 2.804E-07 | 2.222E-11 | 3.7072E-07 | 4.2933E-07 | 1.5572E-07 | 1.3879E-07 | 3.4449E-13 | 5.7475E-09 | 3.7673E-08 | 4.2933E-07 | 2.79E-07 | 5.5405E-12 | 5.7475E-09 | 3.7072E-07 | 4.8672E-07 | 1.9904E-20 | 3.3208E-16 | 2.222E-11 | 8.9109E-11 |
| 5 | 1.3879E-07 | 1.5572E-07 | 4.2933E-07 | 3.7072E-07 | 2.222E-11 | 2.804E-07 | 1.3879E-07 | 2.79E-07 | 4.8672E-07 | 8.9109E-11 | 1.3879E-07 | 1.5572E-07 | 4.2933E-07 | 3.7072E-07 | 2.222E-11 | 4.2933E-07 | 3.7673E-08 | 5.7475E-09 | 3.4449E-13 | 4.8672E-07 | 3.7072E-07 | 5.5405E-12 | 3.3208E-16 |
| 6 | 1.5572E-07 | 1.3879E-07 | 1.5572E-07 | 4.2933E-07 | 3.7072E-07 | 1.3879E-07 | 2.804E-07 | 1.3879E-07 | 2.79E-07 | 4.8672E-07 | 1.5572E-07 | 1.3879E-07 | 1.5572E-07 | 4.2933E-07 | 3.7072E-07 | 2.79E-07 | 4.2933E-07 | 3.7673E-08 | 5.7475E-09 | 3.7072E-07 | 4.8672E-07 | 5.7475E-09 | 5.5405E-12 |
| 7 | 4.2933E-07 | 1.5572E-07 | 1.3879E-07 | 1.5572E-07 | 4.2933E-07 | 2.79E-07 | 1.3879E-07 | 2.804E-07 | 1.3879E-07 | 2.79E-07 | 4.2933E-07 | 1.5572E-07 | 1.3879E-07 | 1.5572E-07 | 4.2933E-07 | 4.2933E-07 | 2.79E-07 | 4.2933E-07 | 3.7673E-08 | 5.7475E-09 | 3.7072E-07 | 3.7072E-07 | 5.7475E-09 |
| 8 | 3.7072E-07 | 4.2933E-07 | 1.5572E-07 | 1.3879E-07 | 1.5572E-07 | 4.8672E-07 | 2.79E-07 | 1.3879E-07 | 2.804E-07 | 1.3879E-07 | 3.7072E-07 | 4.2933E-07 | 1.5572E-07 | 1.3879E-07 | 1.5572E-07 | 3.7673E-08 | 4.2933E-07 | 2.79E-07 | 4.2933E-07 | 5.5405E-12 | 5.7475E-09 | 4.8672E-07 | 3.7072E-07 |
| 9 | 2.222E-11 | 3.7072E-07 | 4.2933E-07 | 1.5572E-07 | 1.3879E-07 | 8.9109E-11 | 4.8672E-07 | 2.79E-07 | 1.3879E-07 | 2.804E-07 | 2.222E-11 | 3.7072E-07 | 4.2933E-07 | 1.5572E-07 | 1.3879E-07 | 5.7475E-09 | 3.7673E-08 | 4.2933E-07 | 2.79E-07 | 3.3208E-16 | 5.5405E-12 | 3.7072E-07 | 4.8672E-07 |
| 10 | 2.79E-07 | 4.2933E-07 | 3.7673E-08 | 5.7475E-09 | 3.4449E-13 | 1.3879E-07 | 1.5572E-07 | 4.2933E-07 | 3.7072E-07 | 2.222E-11 | 2.804E-07 | 1.3879E-07 | 2.79E-07 | 4.8672E-07 | 8.9109E-11 | 1.5572E-07 | 4.2933E-07 | 3.7072E-07 | 2.222E-11 | 2.79E-07 | 4.2933E-07 | 5.7475E-09 | 3.4449E-13 |
| 11 | 4.2933E-07 | 2.79E-07 | 4.2933E-07 | 3.7673E-08 | 5.7475E-09 | 1.5572E-07 | 1.3879E-07 | 1.5572E-07 | 4.2933E-07 | 3.7072E-07 | 1.3879E-07 | 2.804E-07 | 1.3879E-07 | 2.79E-07 | 4.8672E-07 | 1.3879E-07 | 1.5572E-07 | 4.2933E-07 | 3.7072E-07 | 4.2933E-07 | 2.79E-07 | 3.7673E-08 | 5.7475E-09 |
| 12 | 3.7673E-08 | 4.2933E-07 | 2.79E-07 | 4.2933E-07 | 3.7673E-08 | 4.2933E-07 | 1.5572E-07 | 1.3879E-07 | 1.5572E-07 | 4.2933E-07 | 2.79E-07 | 1.3879E-07 | 2.804E-07 | 1.3879E-07 | 2.79E-07 | 1.5572E-07 | 1.3879E-07 | 1.5572E-07 | 4.2933E-07 | 3.7673E-08 | 4.2933E-07 | 4.2933E-07 | 3.7673E-08 |
| 13 | 5.7475E-09 | 3.7673E-08 | 4.2933E-07 | 2.79E-07 | 4.2933E-07 | 3.7072E-07 | 4.2933E-07 | 1.5572E-07 | 1.3879E-07 | 1.5572E-07 | 4.8672E-07 | 2.79E-07 | 1.3879E-07 | 2.804E-07 | 1.3879E-07 | 4.2933E-07 | 1.5572E-07 | 1.3879E-07 | 1.5572E-07 | 5.7475E-09 | 3.7673E-08 | 2.79E-07 | 4.2933E-07 |
| 14 | 3.4449E-13 | 5.7475E-09 | 3.7673E-08 | 4.2933E-07 | 2.79E-07 | 2.222E-11 | 3.7072E-07 | 4.2933E-07 | 1.5572E-07 | 1.3879E-07 | 8.9109E-11 | 4.8672E-07 | 2.79E-07 | 1.3879E-07 | 2.804E-07 | 3.7072E-07 | 4.2933E-07 | 1.5572E-07 | 1.3879E-07 | 3.4449E-13 | 5.7475E-09 | 4.2933E-07 | 2.79E-07 |
| 16 | 3.7072E-07 | 4.8672E-07 | 3.7072E-07 | 5.7475E-09 | 5.5405E-12 | 4.2933E-07 | 2.79E-07 | 4.2933E-07 | 3.7673E-08 | 5.7475E-09 | 1.5572E-07 | 1.3879E-07 | 1.5572E-07 | 4.2933E-07 | 3.7072E-07 | 2.804E-07 | 1.3879E-07 | 2.79E-07 | 4.8672E-07 | 1.5572E-07 | 1.3879E-07 | 4.2933E-07 | 3.7072E-07 |
| 17 | 5.7475E-09 | 3.7072E-07 | 4.8672E-07 | 3.7072E-07 | 5.7475E-09 | 3.7673E-08 | 4.2933E-07 | 2.79E-07 | 4.2933E-07 | 3.7673E-08 | 4.2933E-07 | 1.5572E-07 | 1.3879E-07 | 1.5572E-07 | 4.2933E-07 | 1.3879E-07 | 2.804E-07 | 1.3879E-07 | 2.79E-07 | 4.2933E-07 | 1.5572E-07 | 1.5572E-07 | 4.2933E-07 |
| 18 | 5.5405E-12 | 5.7475E-09 | 3.7072E-07 | 4.8672E-07 | 3.7072E-07 | 5.7475E-09 | 3.7673E-08 | 4.2933E-07 | 2.79E-07 | 4.2933E-07 | 3.7072E-07 | 4.2933E-07 | 1.5572E-07 | 1.3879E-07 | 1.5572E-07 | 2.79E-07 | 1.3879E-07 | 2.804E-07 | 1.3879E-07 | 3.7072E-07 | 4.2933E-07 | 1.3879E-07 | 1.5572E-07 |
| 19 | 3.3208E-16 | 5.5405E-12 | 5.7475E-09 | 3.7072E-07 | 4.8672E-07 | 3.4449E-13 | 5.7475E-09 | 3.7673E-08 | 4.2933E-07 | 2.79E-07 | 2.222E-11 | 3.7072E-07 | 4.2933E-07 | 1.5572E-07 | 1.3879E-07 | 4.8672E-07 | 2.79E-07 | 1.3879E-07 | 2.804E-07 | 2.222E-11 | 3.7072E-07 | 1.5572E-07 | 1.3879E-07 |
| 20 | 8.9109E-11 | 2.222E-11 | 3.4449E-13 | 3.3208E-16 | 1.9904E-20 | 4.8672E-07 | 3.7072E-07 | 5.7475E-09 | 5.5405E-12 | 3.3208E-16 | 2.79E-07 | 4.2933E-07 | 3.7673E-08 | 5.7475E-09 | 3.4449E-13 | 1.5572E-07 | 4.2933E-07 | 3.7072E-07 | 2.222E-11 | 2.804E-07 | 1.3879E-07 | 4.8672E-07 | 8.9109E-11 |
| 21 | 2.222E-11 | 8.9109E-11 | 2.222E-11 | 3.4449E-13 | 3.3208E-16 | 3.7072E-07 | 4.8672E-07 | 3.7072E-07 | 5.7475E-09 | 5.5405E-12 | 4.2933E-07 | 2.79E-07 | 4.2933E-07 | 3.7673E-08 | 5.7475E-09 | 1.3879E-07 | 1.5572E-07 | 4.2933E-07 | 3.7072E-07 | 1.3879E-07 | 2.804E-07 | 2.79E-07 | 4.8672E-07 |
| 23 | 3.3208E-16 | 3.4449E-13 | 2.222E-11 | 8.9109E-11 | 2.222E-11 | 5.5405E-12 | 5.7475E-09 | 3.7072E-07 | 4.8672E-07 | 3.7072E-07 | 5.7475E-09 | 3.7673E-08 | 4.2933E-07 | 2.79E-07 | 4.2933E-07 | 4.2933E-07 | 1.5572E-07 | 1.3879E-07 | 1.5572E-07 | 4.8672E-07 | 2.79E-07 | 2.804E-07 | 1.3879E-07 |
| 24 | 1.9904E-20 | 3.3208E-16 | 3.4449E-13 | 2.222E-11 | 8.9109E-11 | 3.3208E-16 | 5.5405E-12 | 5.7475E-09 | 3.7072E-07 | 4.8672E-07 | 3.4449E-13 | 5.7475E-09 | 3.7673E-08 | 4.2933E-07 | 2.79E-07 | 3.7072E-07 | 4.2933E-07 | 1.5572E-07 | 1.3879E-07 | 8.9109E-11 | 4.8672E-07 | 1.3879E-07 | 2.804E-07 |

0.000001 is as good as it gets since I am printing 6 digits after the decimal point, and usually errors would become obvious. I could print more digits, but it does not matter usually bugs are not this accurate.

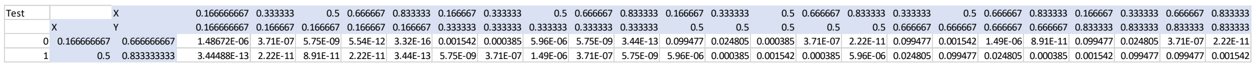
Next, I compare the $k_*^T$. I had a bug in using wrong indices, but thanks to the excel file I managed to fix it early. In fact, this was the last stage before I run this testing. Overall, in all previous steps I ran a small test checking few numbers to make sure things are good, but I decided to check $k_*^T$ with everything else and that is why I could not catch the bug earlier. I went an ran my code for step 5 again to update the running time numbers. I have included the result after correcting the bug in here:

$k_*^T$ result printed by code (partial view):



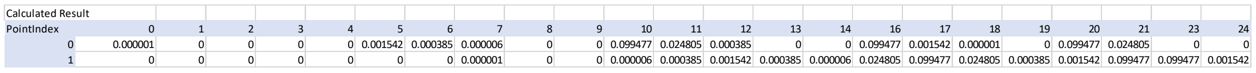
```
trainingK after noise:
  0.000001  0.000000  0.000000  0.000000  0.000000  0.001542  0.000385  0.000006  0.000000  0.000000  0.099477  0.024805  0.000385  0.000000  0.000000  0.099477  0.001542  0.00
  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000001  0.000000  0.000000  0.000006  0.000385  0.001542  0.000385  0.000006  0.024805  0.099477  0.02
```

The result of $k_*^T$ computed on Excel:



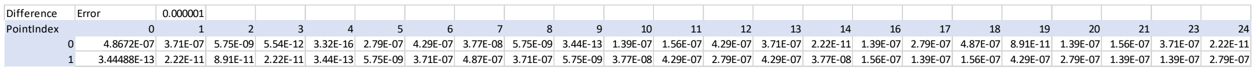
| Test | | X | 0.166666667 | 0.333333 | 0.5 | 0.666667 | 0.833333 | 0.166667 | 0.333333 | 0.5 | 0.666667 | 0.833333 | 0.166667 | 0.333333 | 0.5 | 0.666667 | 0.833333 | 0.333333 | 0.5 | 0.666667 | 0.833333 | 0.166667 | 0.333333 | 0.666667 | 0.833333 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | X | Y | 0.166666667 | 0.166667 | 0.166667 | 0.166667 | 0.166667 | 0.333333 | 0.333333 | 0.333333 | 0.333333 | 0.333333 | 0.5 | 0.5 | 0.5 | 0.5 | 0.5 | 0.666667 | 0.666667 | 0.666667 | 0.666667 | 0.833333 | 0.833333 | 0.833333 | 0.833333 |
| 0 | 0.166666667 | 0.666666667 | 1.48672E-06 | 3.71E-07 | 5.75E-09 | 5.54E-12 | 3.32E-16 | 0.001542 | 0.000385 | 5.96E-06 | 5.75E-09 | 3.44E-13 | 0.099477 | 0.024805 | 0.000385 | 3.71E-07 | 2.22E-11 | 0.099477 | 0.001542 | 1.49E-06 | 8.91E-11 | 0.099477 | 0.024805 | 3.71E-07 | 2.22E-11 |
| 1 | 0.5 | 0.833333333 | 3.44488E-13 | 2.22E-11 | 8.91E-11 | 2.22E-11 | 3.44E-13 | 5.75E-09 | 3.71E-07 | 1.49E-06 | 3.71E-07 | 5.75E-09 | 5.96E-06 | 0.000385 | 0.001542 | 0.000385 | 5.96E-06 | 0.024805 | 0.099477 | 0.024805 | 0.000385 | 0.001542 | 0.099477 | 0.099477 | 0.001542 |

The result of $k_*^T$ computed by my code:



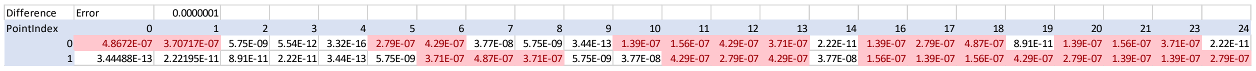
| Calculated Result | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| PointIndex | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 16 | 17 | 18 | 19 | 20 | 21 | 23 | 24 |
| 0 | 0.000001 | 0 | 0 | 0 | 0 | 0.001542 | 0.000385 | 0.000006 | 0 | 0 | 0.099477 | 0.024805 | 0.000385 | 0 | 0 | 0.099477 | 0.001542 | 0.000001 | 0 | 0.099477 | 0.024805 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0.000001 | 0 | 0 | 0.000006 | 0.000385 | 0.001542 | 0.000385 | 0.000006 | 0.024805 | 0.099477 | 0.024805 | 0.000385 | 0.001542 | 0.099477 | 0.099477 | 0.001542 |

$k_*^T$, absolute error between my code and excel calculation (error above 0.000001 is turned red): No error above 0.000001 since there are no red cells



| Difference | Error | 0.000001 | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| PointIndex | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 16 | 17 | 18 | 19 | 20 | 21 | 23 | 24 |
| 0 | 4.8672E-07 | 3.71E-07 | 5.75E-09 | 5.54E-12 | 3.32E-16 | 2.79E-07 | 4.29E-07 | 3.77E-08 | 5.75E-09 | 3.44E-13 | 1.39E-07 | 1.56E-07 | 4.29E-07 | 3.71E-07 | 2.22E-11 | 1.39E-07 | 2.79E-07 | 4.87E-07 | 8.91E-11 | 1.39E-07 | 1.56E-07 | 3.71E-07 | 2.22E-11 |
| 1 | 3.44488E-13 | 2.22E-11 | 8.91E-11 | 2.22E-11 | 3.44E-13 | 5.75E-09 | 3.71E-07 | 4.87E-07 | 3.71E-07 | 5.75E-09 | 3.77E-08 | 4.29E-07 | 2.79E-07 | 4.29E-07 | 3.77E-08 | 1.56E-07 | 1.39E-07 | 1.56E-07 | 4.29E-07 | 2.79E-07 | 1.39E-07 | 1.39E-07 | 2.79E-07 |

$k_*^T$ before noise, absolute error between my code and excel calculation (error above 0.0000001 is turned red): many cells turned red showing error above 0.0000001

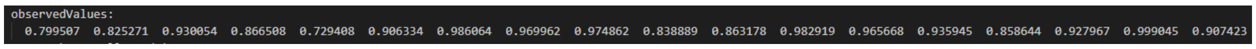


| Difference | Error | 0.0000001 | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| PointIndex | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 16 | 17 | 18 | 19 | 20 | 21 | 23 | 24 |
| 0 | 4.8672E-07 | 3.70717E-07 | 5.75E-09 | 5.54E-12 | 3.32E-16 | 2.79E-07 | 4.29E-07 | 3.77E-08 | 5.75E-09 | 3.44E-13 | 1.39E-07 | 1.56E-07 | 4.29E-07 | 3.71E-07 | 2.22E-11 | 1.39E-07 | 2.79E-07 | 4.87E-07 | 8.91E-11 | 1.39E-07 | 1.56E-07 | 3.71E-07 | 2.22E-11 |
| 1 | 3.44488E-13 | 2.22195E-11 | 8.91E-11 | 2.22E-11 | 3.44E-13 | 5.75E-09 | 3.71E-07 | 4.87E-07 | 3.71E-07 | 5.75E-09 | 3.77E-08 | 4.29E-07 | 2.79E-07 | 4.29E-07 | 3.77E-08 | 1.56E-07 | 1.39E-07 | 1.56E-07 | 4.29E-07 | 2.79E-07 | 1.39E-07 | 1.39E-07 | 2.79E-07 |

0.000001 is as good as it gets since I am printing 6 digits after the decimal point, and usually errors would become obvious. I could print more digits, but it does not matter usually bugs are not this accurate.

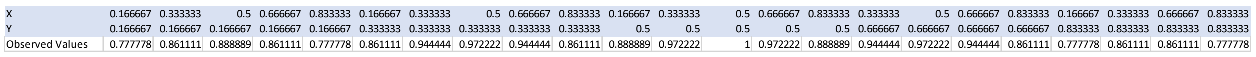
Lastly, I checked the observed values for the train point. I have tested this part in minor project, so an inspection was carried out by eyes.

Observed values printed by my code (take note that they have random noise in the range ±0.05):



```
observedValues:
  0.799507  0.825271  0.930054  0.866508  0.729408  0.906334  0.986064  0.969962  0.974862  0.838889  0.863178  0.982919  0.965668  0.935945  0.858644  0.927967  0.999045  0.907423
```

Observed values computed on excel without noise:



| X | 0.166667 | 0.333333 | 0.5 | 0.666667 | 0.833333 | 0.166667 | 0.333333 | 0.5 | 0.666667 | 0.833333 | 0.166667 | 0.333333 | 0.5 | 0.666667 | 0.833333 | 0.333333 | 0.5 | 0.666667 | 0.833333 | 0.166667 | 0.333333 | 0.666667 | 0.833333 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Y | 0.166667 | 0.166667 | 0.166667 | 0.166667 | 0.166667 | 0.333333 | 0.333333 | 0.333333 | 0.333333 | 0.333333 | 0.5 | 0.5 | 0.5 | 0.5 | 0.5 | 0.666667 | 0.666667 | 0.666667 | 0.666667 | 0.833333 | 0.833333 | 0.833333 | 0.833333 |
| Observed Values | 0.777778 | 0.861111 | 0.888889 | 0.861111 | 0.777778 | 0.861111 | 0.944444 | 0.972222 | 0.944444 | 0.861111 | 0.888889 | 0.972222 | 1 | 0.972222 | 0.888889 | 0.944444 | 0.972222 | 0.944444 | 0.861111 | 0.777778 | 0.861111 | 0.861111 | 0.777778 |

Part 7) Running time single vs multiple parallel region

For everything developed so far, I decided to run a test, in this test I placed everything in the single region and once I did not have a single region and instead multiple parallel regions were created. I wanted to check if having OpenMP active at all time, makes it run better. The table below shows the result. As it can be shown in the table it is better not to run everything in a single, this is mainly because for declaring a parallel for loop you must create another parallel region. After running the two case, I decided to run another test where I use one parallel region and multiple single regions. In this last case, for loops are in parallel region after a single region. Single regions synchronize at the end so this should work fine in terms of code correctness. As it can be shown these syncing processes are even more time consuming. After running these tests, it is apparent that having the parallel region whenever needed is the best approach.

| Approach\m size | m = 50 | m = 100 | m = 150 |
|---|---|---|---|
| Parallel Time, with single (ms) | 61.720 | 798.608 | 3,966.425 |
| Parallel Time, without single (ms) | 15.853 | 43.830 | 168.090 |
| Parallel Time, multiple single (ms) | 67.944 | 688.529 | 3,475.340 |

Part 8) Adding LU factorization, substitution and multiplication from minor project

I added LU factorization, substitution and final multiplication to my project from minor project. To keep this report short, I just report the performance numbers. In minor project I did more analysis of the running time. The following table shows the time it takes before multiplication.

| Approach\m size | m = 50 | m = 100 | m = 150 |
|---|---|---|---|
| Parallel Time everything upto part 8 (ms) | 147.835 | 50,356.402 | Very long Omitted |

In minor project, I have tested these parts, but I decided to test again to make sure having multi value estimation works correctly. I used https://matrixcalc.org/en/ for my testing. This website computes both matrix inverse and LU factorization. To speed up my testing I removed all the noises from my code and performed an inverse on K matrix. Then multiplied it with f which I calculated and tested before. I then took the result and multiplied to $k_*^T$. I got the same results as my code:

L1 = L2 = 0.1

| Approach\ Estimation for | X = 0.166667, Y = 0.666667 | X = 0.500000, Y = 0.833333 |
|---|---|---|
| My Code's Estimation | 0.486719 | 0.473991 |
| Matrix Testing Website's Calculation | 0.486719 | 0.473991 |
| Actual Value Based on function f | 0.861111 | 0.888889 |

As it can be seen my code has computed correct LU factorization, substitution and multiplication; however, the actual value is significantly different from expected value. This may be due to bad L1 a L2 value and the small value of m. We will see how things workout later as we compare different values of L1 and L2.

Part 9) Mean Square Error

For MSE calculation I used OpenMp's reduction and check the result with a serial version of the code and also an excel version. However, after doing so I decided to run a test to see how long it takes. The table below shows the time required for computing MSE except for its last division.

| Approach\m size | m = 5, 2 points | m = 50, 250 points | m = 100, 1000 points |
|---|---|---|---|
| Serial Time (us) | 1 | 1 | 3 |
| Parallel Time, Second test (us) | 259 | 116 | 156 |

One more time it seams that the overhead for parallelizing is way larger than the actual time to compute the whole thing in serial. Still one mystery is the amount of time the parallel version needs for smaller loops. In this test OpenMP has been running prior to calling this function but still this behavior is observed, Maybe OpenMP is especially inefficient for when the task is smaller than the number of available threads. Definitely a loop with 2 iteration is lower than 28 thread (OMP_NUM_THREADS=28) set for OpenMP. In conclusion this step is serial too.

Part 10) Putting everything together

So I added 2 loops that iterate through L1 and L2 values. Based on the assignment I assume that L1 and L2 values we should test are between [0.1, 1.0] with 0.1 increments. I made this assumption because otherwise the getting the inputs to the code would become very difficult. Another part that I had to test in this part was testing the minimum finder for MSE values. I am 100% certain that the parallel version takes longer because it has even less computation than part 9 and there are only 100 values to compare. So I just tested if my serial version works correctly. The table below shows MSE values for some m values.

| Approach\m size | m = 5, 2 test points | m = 25, 62 test points | m = 50, 250 test points |
|---|---|---|---|
| MSE error | 0.000517 | 0.000033 | 0.000012 |
| Time (s) with some extra value printing | 0.054082 | 0.534758 | 12.128539 |
| L1 | 0.1 | 0.1 | 0.2 |
| L2 | 0.7 | 0.9 | 0.1 |

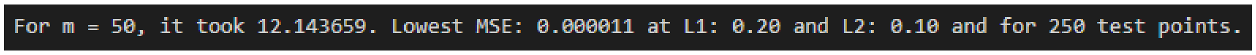
In final test I decided to parallel the loop iterating L1 and L2. The table below shows the run time. Take note that I have removed all the printing related tasks and that is why it runs faster than the previous table in serial.

| Approach\m size | m = 5, 2 test points | m = 25, 62 test points | m = 50, 250 test points |
|---|---|---|---|
| Serial Time (ms) | 54.715 | 529.780 | 11,909.745 |
| Parallel Time, 1 loop (ms) | 11.345 | 1,026.361 | 46,271.799 |
| Parallel Time, 2 loop (ms) | 15.385 | 559.771 | 20,424.408 |

In red cells the MSE value was NAN. I think this is due to creating many parallel threads. Possibly OpenMP has a higher threshold. To test this, I decided to Serialize everything in my code and instead paradelike the L1 and L2. I think this would result in much faster performance. In parallel part of the table below only the L1 and L2 loops are parallelized and no part inside them is parallelized.

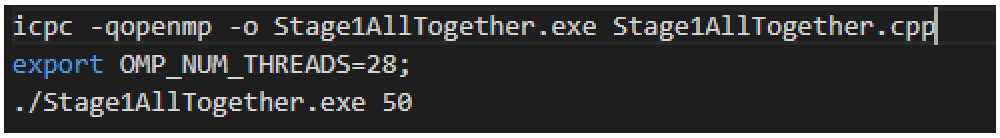
| Approach\m size | m = 5, 2 test points | m = 25, 62 test points | m = 50, 250 test points |
|---|---|---|---|
| Serial Time (ms) | 54.715 | 529.780 | 11,909.745 |
| Parallel Time, 1 loop (ms) | 11.959 | 992.278 | 45,582.864 |
| Parallel Time, 2 loop (ms) | 12.692 | 609.356 | 19,501.417 |

I am not sure why again the red cells contained as their MSE NAN. I wrap it up with L1 and L2 being serial, because even if the MSE value's is ignored, the parallel solutions take longer. With this I conclude my project, I have attached my excel files as well and the output format of my code looks like this:


```
For m = 50, it took 12.143659. Lowest MSE: 0.000011 at L1: 0.20 and L2: 0.10 and for 250 test points.
```

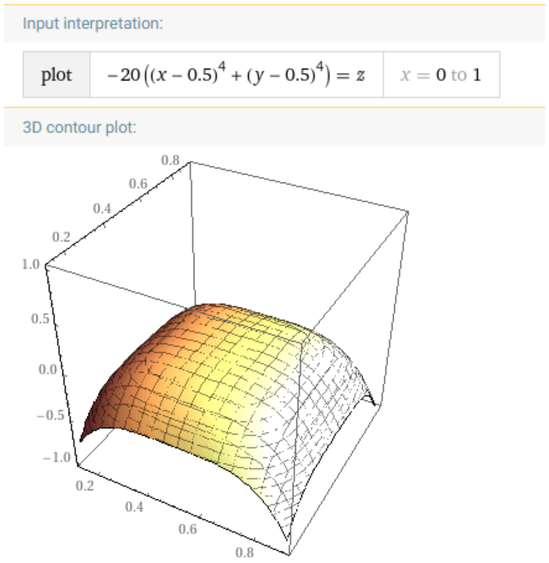
In the following image you can see how to compile my program and run it with m = 50. Take note that I am automatically using the function in part 3.

The function is $f(x_i, y_i) = 1 - ((x_i - 0.5)^2 + (y_i - 0.5)^2) + d_{ij}$


```
icpc -qopenmp -o Stage1AllTogether.exe Stage1AllTogether.cpp
export OMP_NUM_THREADS=28;
./Stage1AllTogether.exe 50
```

3. (5 points) Apply your code to another data set to see how it performs. You may choose any appropriate data set that you can find.

I was planning to use my code to predict the surface of mountains or other objects that have been scanned but I could not find any data that I could easily use, so I decided to make a mountain looking surface and change the equation in my code. While doing so I thought of reconstructing an image using my code, but with C++ this becomes difficult because it does not have a straightforward approach to open images and save them with its standard function. So, I decided to use my mountain for it.



Source: https://www.wolframalpha.com/input/?i=-20*%28%28x+-+0.5%29%5E4+%2B+%28y+-+.5%29%5E4%29++%3D+z+from+0+to+1

$$f(x_i, y_i) = -20\,((x_i - 0.5)^4 + (y_i - 0.5)^4) + d_{ij}$$

Where $d_{ij}$ is noise ranging from [-0.05, +0.05].

| Approach\m size | m = 5, 2 test points | m = 25, 62 test points | m = 50, 250 test points |
|---|---|---|---|
| MSE error | 0.006889 | 0.000406 | 0.000118 |
| Time (s) | 0.023923 | 0.452031 | 12.503198 |
| L1 | 1.0 | 0.1 | 0.1 |
| L2 | 1.0 | 0.1 | 0.1 |

As expected, the MSE error is higher because my plane changes very quickly towards the corners but previous function is slowly changing.