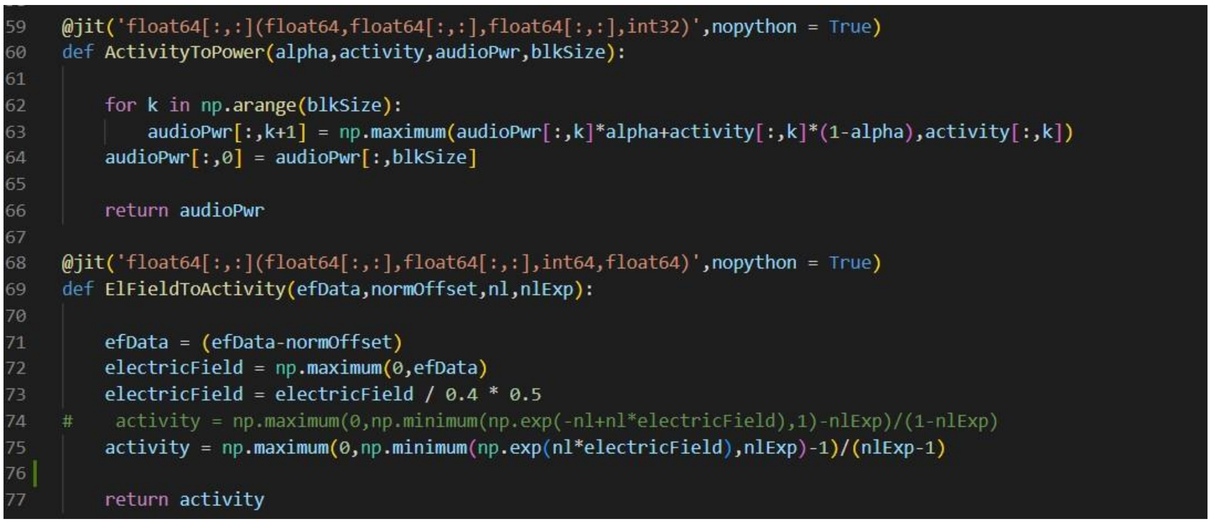


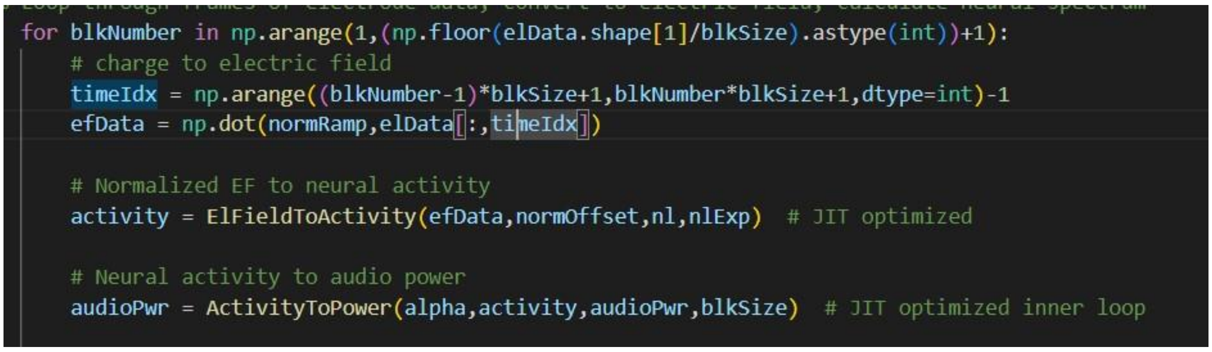
```python
@jit('float64[:,:](float64,float64[:,:],float64[:,:],int32)',nopython = True)
def ActivityToPower(alpha,activity,audioPwr,blkSize):

    for k in np.arange(blkSize):
        audioPwr[:,k+1] = np.maximum(audioPwr[:,k]*alpha+activity[:,k]*(1-alpha),activity[:,k])
    audioPwr[:,0] = audioPwr[:,blkSize]

    return audioPwr

@jit('float64[:,:](float64[:,:],float64[:,:],int64,float64)',nopython = True)
def ElFieldToActivity(efData,normOffset,nl,nlExp):

    efData = (efData-normOffset)
    electricField = np.maximum(0,efData)
    electricField = electricField / 0.4 * 0.5
#    activity = np.maximum(0,np.minimum(np.exp(-nl+nl*electricField),1)-nlExp)/(1-nlExp)
    activity = np.maximum(0,np.minimum(np.exp(nl*electricField),nlExp)-1)/(nlExp-1)

    return activity
```

Figure 3. Python code of converting functions.

2. For each accelerated function, provide a brief summary of any investigations you conducted to arrive at your choice of pragma(s) used to implement the function into the PL. Include screen capture of report output(s) showing comparison of speed of different implementations as well as PL resources used.

- **PROCEDURE**
  a) Export the variables 'data

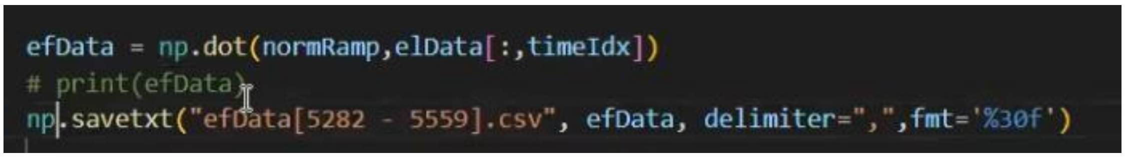
Firstly, I want to demonstrate how I convert the above functions into C before showing the choosing pragmas. Some input variables need to export the data from MATLAB or Python. While the list of input variables that need to export the data are: blkNum, TimeIdx, normOffset, efData, the output variables are activity, audioPwr. The non-mentioned variables are assigned value variables.



```python
for blkNumber in np.arange(1,(np.floor(elData.shape[1]/blkSize).astype(int))+1):
    # charge to electric field
    timeIdx = np.arange((blkNumber-1)*blkSize+1,blkNumber*blkSize+1,dtype=int)-1
    efData = np.dot(normRamp,elData[:,timeIdx])

    # Normalized EF to neural activity
    activity = ElFieldToActivity(efData,normOffset,nl,nlExp)  # JIT optimized

    # Neural activity to audio power
    audioPwr = ActivityToPower(alpha,activity,audioPwr,blkSize)  # JIT optimized inner loop
```

Figure 3. List of variables that need to export the data

For blknum, the range of this variable is from 1 to 1543. The term "np.floor" means that if the output number is not an integer number, it will take an integer number below that one [1] and then plus 1. Blksize =287, which was exported from MATLAB. For the TimeIdx variable, I chose blknum = 20 because, with the small value of blknum, I saw that the TimeIdx in CSV file seemed to be all 0. With the value of blknum = 20, the data was different from 0, which is from 5282 to 5559. With the data that I had; I was able to export the efData variable.

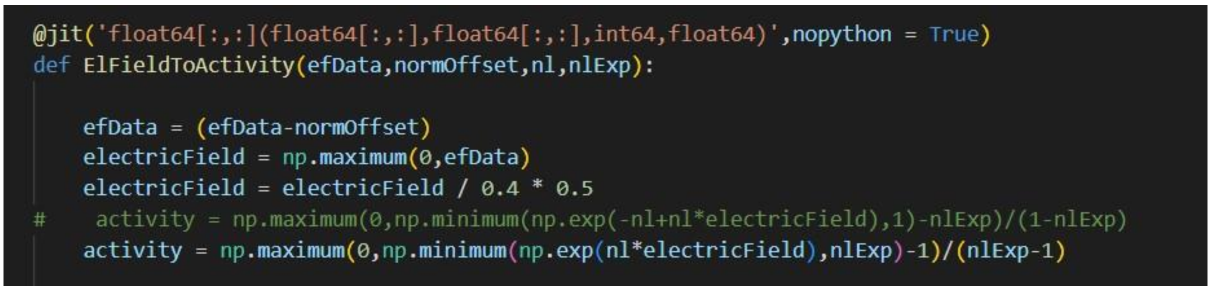


```
efData = np.dot(normRamp,elData[:,timeIdx])
# print(efData)
np.savetxt("efData[5282 - 5559].csv", efData, delimiter=",",fmt='%30f')
```

Figure 4. Exporting efData in CSV

The purpose of using %30f is to convert exponential numbers to decimal numbers.

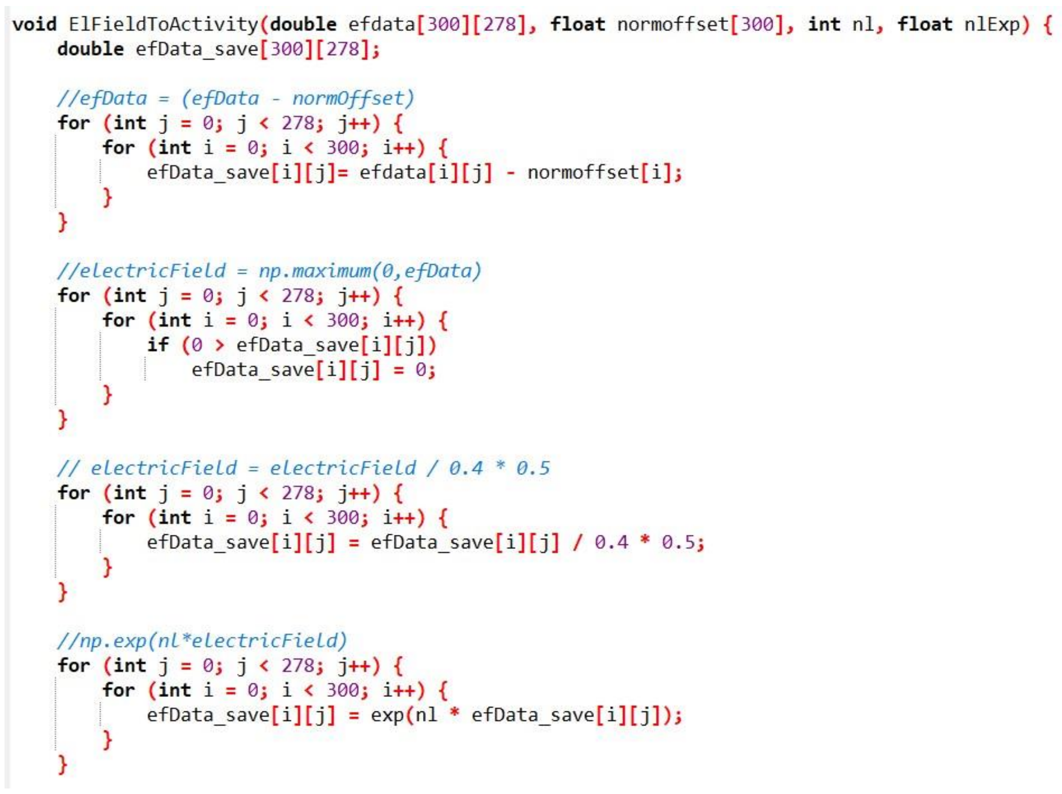
b) Converting Python to C



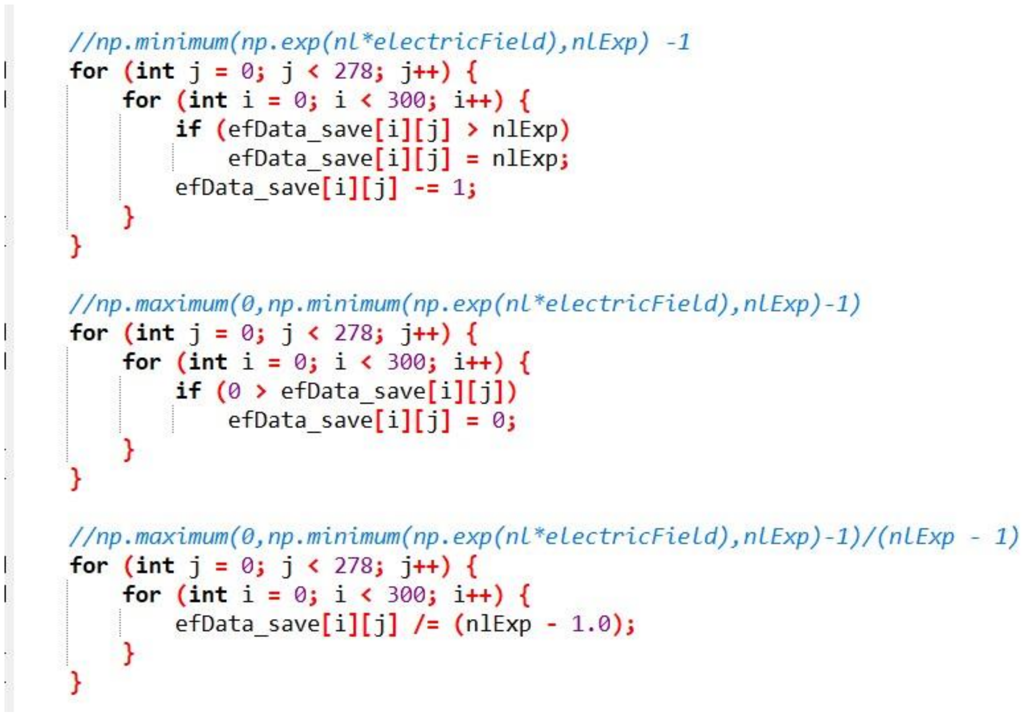
```
@jit('float64[:,:](float64[:,:],float64[:,:],int64,float64)',nopython = True)
def ElFieldToActivity(efData,normOffset,nl,nlExp):

    efData = (efData-normOffset)
    electricField = np.maximum(0,efData)
    electricField = electricField / 0.4 * 0.5
#    activity = np.maximum(0,np.minimum(np.exp(-nl+nl*electricField),1)-nlExp)/(1-nlExp)
    activity = np.maximum(0,np.minimum(np.exp(nl*electricField),nlExp)-1)/(nlExp-1)
```

Figure 5. ElFieldToActivity Python Code

The ElFieldToActivity code in C below is used for identical blknum, which is equal to 20. The code can be used with different values of blknum by changing the assigned value.



```c
void ElFieldToActivity(double efdata[300][278], float normoffset[300], int nl, float nlExp) {
    double efData_save[300][278];

    //efData = (efData - normOffset)
    for (int j = 0; j < 278; j++) {
        for (int i = 0; i < 300; i++) {
            efData_save[i][j]= efdata[i][j] - normoffset[i];
        }
    }

    //electricField = np.maximum(0,efData)
    for (int j = 0; j < 278; j++) {
        for (int i = 0; i < 300; i++) {
            if (0 > efData_save[i][j])
                efData_save[i][j] = 0;
        }
    }

    // electricField = electricField / 0.4 * 0.5
    for (int j = 0; j < 278; j++) {
        for (int i = 0; i < 300; i++) {
            efData_save[i][j] = efData_save[i][j] / 0.4 * 0.5;
        }
    }

    //np.exp(nl*electricField)
    for (int j = 0; j < 278; j++) {
        for (int i = 0; i < 300; i++) {
            efData_save[i][j] = exp(nl * efData_save[i][j]);
        }
    }
```

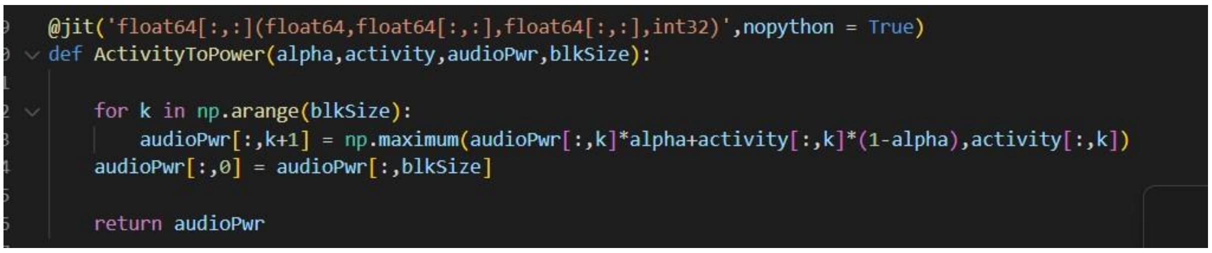


```c
    //np.minimum(np.exp(nl*electricField),nlExp) -1
    for (int j = 0; j < 278; j++) {
        for (int i = 0; i < 300; i++) {
            if (efData_save[i][j] > nlExp)
                efData_save[i][j] = nlExp;
            efData_save[i][j] -= 1;
        }
    }

    //np.maximum(0,np.minimum(np.exp(nl*electricField),nlExp)-1)
    for (int j = 0; j < 278; j++) {
        for (int i = 0; i < 300; i++) {
            if (0 > efData_save[i][j])
                efData_save[i][j] = 0;
        }
    }

    //np.maximum(0,np.minimum(np.exp(nl*electricField),nlExp)-1)/(nlExp - 1)
    for (int j = 0; j < 278; j++) {
        for (int i = 0; i < 300; i++) {
            efData_save[i][j] /= (nlExp - 1.0);
        }
    }
```

Figure 6. The converting C code of ElFieldToActivity

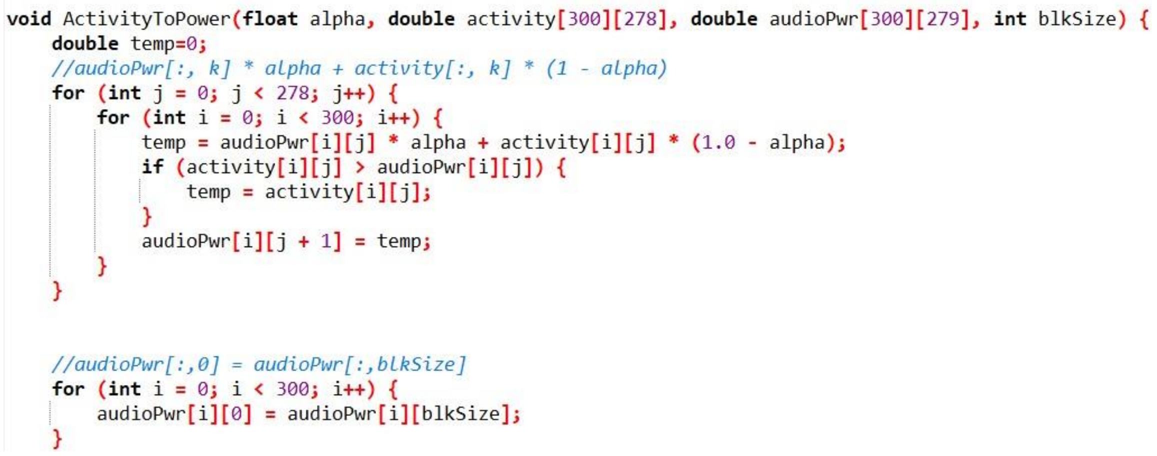
As can be seen from the above figure is that there are some terms in Python that need to identify. While the term” np.maximum” will choose the maximum element in all arrays and store it in the new one [2][3], the term “ np.minimum” will select the minimum value in all arrays and store it into the new array [4].

The purpose of using the loop for Row before the loop for Column is that when exported the NormOffset data, the size was 300x1. Therefore, the efData_Save matrix needs to be inverted so it can be able to minus the normOffset.



```
@jit('float64[:,:](float64,float64[:,:],float64[:,:],int32)',nopython = True)
def ActivityToPower(alpha,activity,audioPwr,blkSize):

    for k in np.arange(blkSize):
        audioPwr[:,k+1] = np.maximum(audioPwr[:,k]*alpha+activity[:,k]*(1-alpha),activity[:,k])
    audioPwr[:,0] = audioPwr[:,blkSize]

    return audioPwr
```

Figure 7. ActivityToPower Python Code



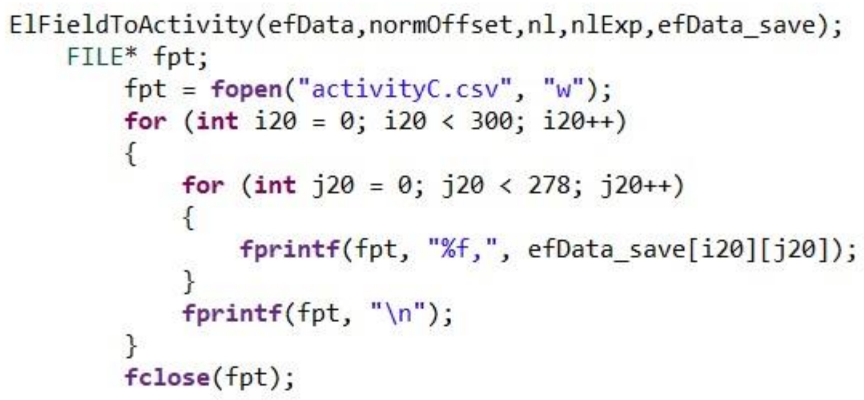
```
void ActivityToPower(float alpha, double activity[300][278], double audioPwr[300][279], int blkSize) {
    double temp=0;
    //audioPwr[:, k] * alpha + activity[:, k] * (1 - alpha)
    for (int j = 0; j < 278; j++) {
        for (int i = 0; i < 300; i++) {
            temp = audioPwr[i][j] * alpha + activity[i][j] * (1.0 - alpha);
            if (activity[i][j] > audioPwr[i][j]) {
                temp = activity[i][j];
            }
            audioPwr[i][j + 1] = temp;
        }
    }


    //audioPwr[:,0] = audioPwr[:,blkSize]
    for (int i = 0; i < 300; i++) {
        audioPwr[i][0] = audioPwr[i][blkSize];
    }
```

Figure 8. ActivityToPower C Code

The ActivitytoPower code in C below is used for identical blknum, which is equal to 20. The code can be used with different values of blknum by changing the assigned value. The size of the audioPwr matrix is 300 x 279 because based on the Python code, line 207 in the Vocoder function, I saw that the size was equal to Blksize +1, whereas blksize = 278.

c) Test Bench

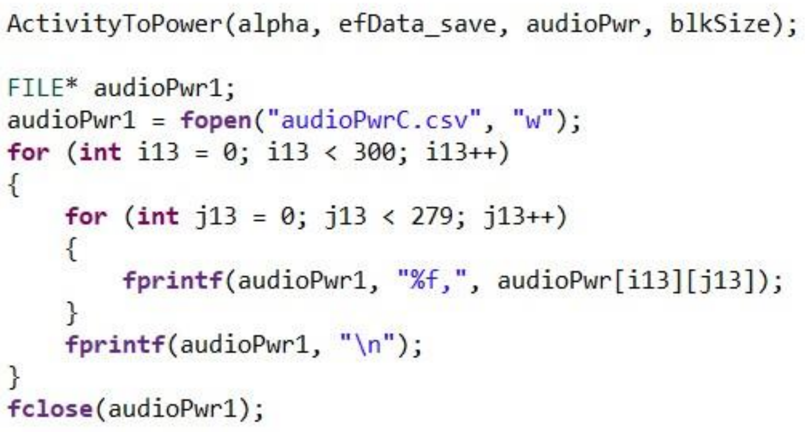
The test bench is implemented to compare the output after execution and the original output from Python to check the correctness of the algorithm after converting it from Python to C. After executing the ElfieldToActivity function, the output will be saved in a CSV file for later comparison purposes, which is shown in the below figure.



```
ElFieldToActivity(efData,normOffset,nl,nlExp,efData_save);
    FILE* fpt;
        fpt = fopen("activityC.csv", "w");
        for (int i20 = 0; i20 < 300; i20++)
        {
            for (int j20 = 0; j20 < 278; j20++)
            {
                fprintf(fpt, "%f,", efData_save[i20][j20]);
            }
            fprintf(fpt, "\n");
        }
        fclose(fpt);
```

Figure 9. import the output to the CSV file

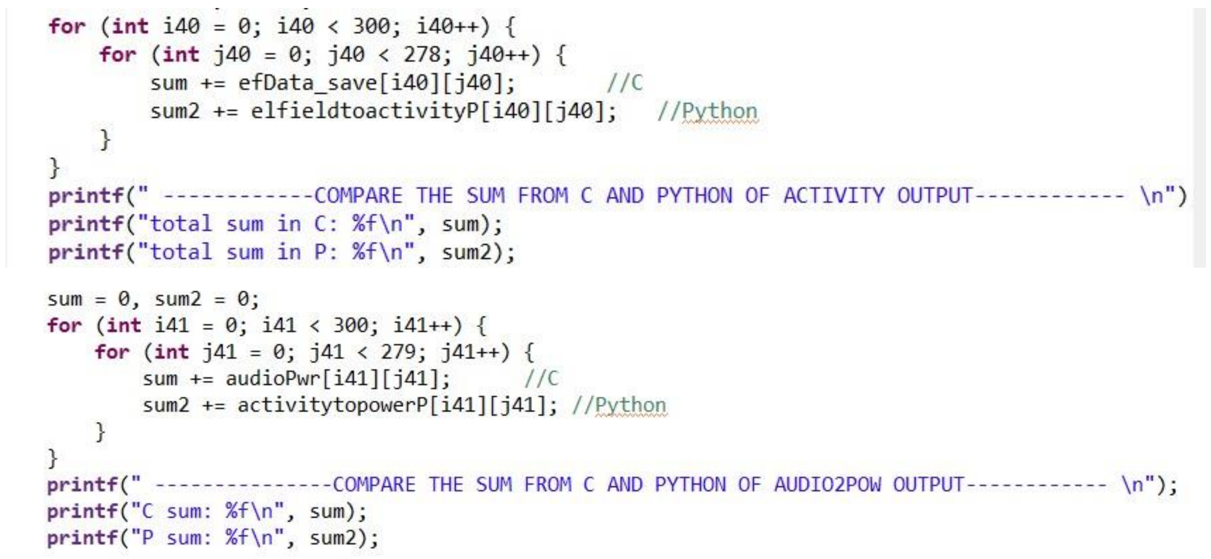
The same procedure is used for the ActivityToPower function.



```
ActivityToPower(alpha, efData_save, audioPwr, blkSize);

FILE* audioPwr1;
audioPwr1 = fopen("audioPwrC.csv", "w");
for (int i13 = 0; i13 < 300; i13++)
{
    for (int j13 = 0; j13 < 279; j13++)
    {
        fprintf(audioPwr1, "%f,", audioPwr[i13][j13]);
    }
    fprintf(audioPwr1, "\n");
}
fclose(audioPwr1);
```

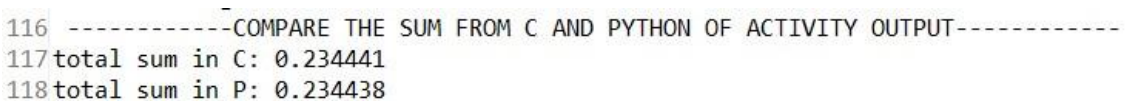
Figure 10. import the output to a CSV file

There are 2 test cases that I used to test the correctness of the implementing code. Firstly, I will check the sum of the output from C and the output of Python.



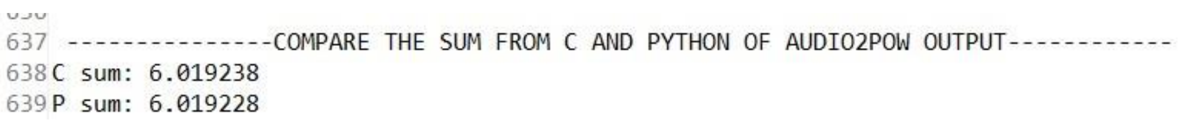
```
for (int i40 = 0; i40 < 300; i40++) {
    for (int j40 = 0; j40 < 278; j40++) {
        sum += efData_save[i40][j40];       //C
        sum2 += elfieldtoactivityP[i40][j40];   //Python
    }
}
printf(" ------------COMPARE THE SUM FROM C AND PYTHON OF ACTIVITY OUTPUT------------ \n")
printf("total sum in C: %f\n", sum);
printf("total sum in P: %f\n", sum2);

sum = 0, sum2 = 0;
for (int i41 = 0; i41 < 300; i41++) {
    for (int j41 = 0; j41 < 279; j41++) {
        sum += audioPwr[i41][j41];      //C
        sum2 += activitytopowerP[i41][j41]; //Python
    }
}
printf(" ---------------COMPARE THE SUM FROM C AND PYTHON OF AUDIO2POW OUTPUT------------ \n");
printf("C sum: %f\n", sum);
printf("P sum: %f\n", sum2);
```

Figure 11. calculate and compare the sum of both files



```
 ------------COMPARE THE SUM FROM C AND PYTHON OF ACTIVITY OUTPUT------------
total sum in C: 0.234441
total sum in P: 0.234438
```

Figure 12. The output of both files in ElfieldToActivity

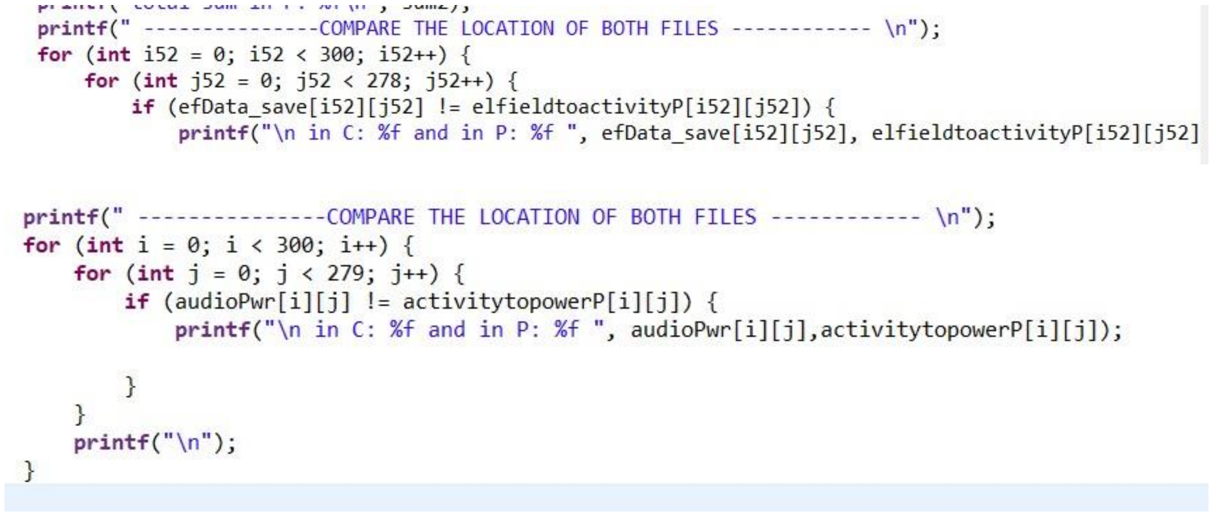


```
 ---------------COMPARE THE SUM FROM C AND PYTHON OF AUDIO2POW OUTPUT------------
C sum: 6.019238
P sum: 6.019228
```

Figure 13. The output of both files in ActivityToPower

It can be seen from Figures 12 and 13 that the sum output in C is slightly higher than the original output sum from Python. The difference in those sums occurs due to the tolerance and byte limitation of the variable type.

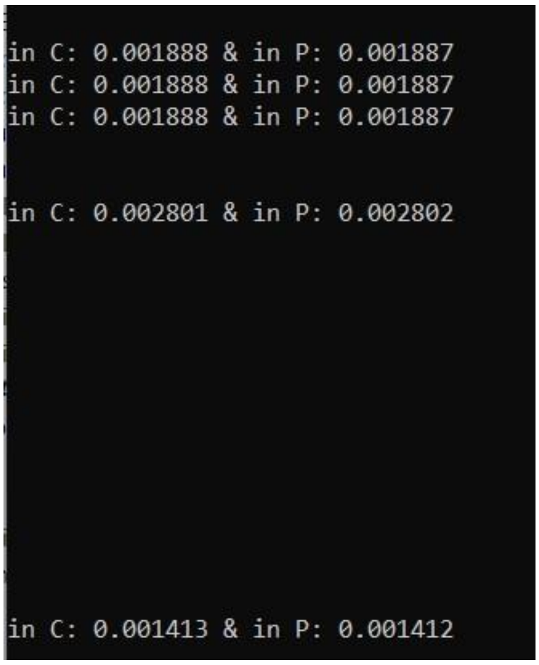
The second test case is to justify why there is a difference between those files.



```
printf(" ---------------COMPARE THE LOCATION OF BOTH FILES ------------ \n");
for (int i52 = 0; i52 < 300; i52++) {
    for (int j52 = 0; j52 < 278; j52++) {
        if (efData_save[i52][j52] != elfieldtoactivityP[i52][j52]) {
            printf("\n in C: %f and in P: %f ", efData_save[i52][j52], elfieldtoactivityP[i52][j52]
```

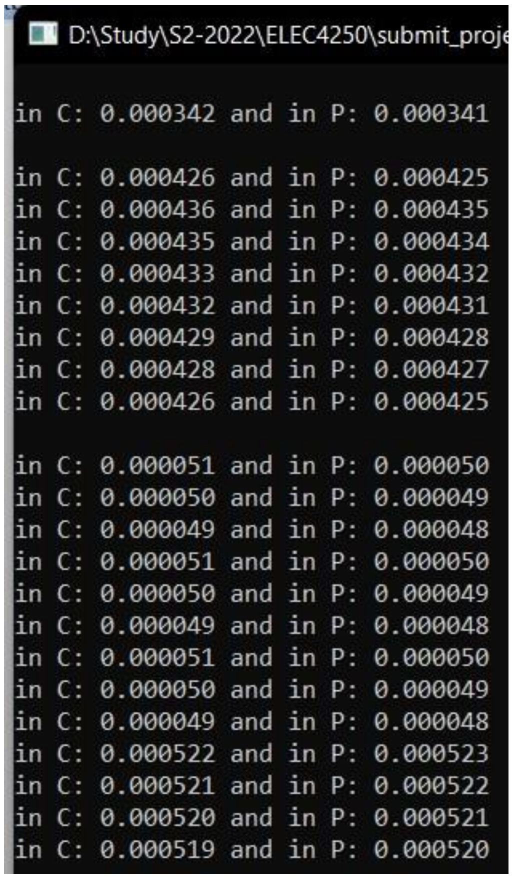
```
printf(" ---------------COMPARE THE LOCATION OF BOTH FILES ------------ \n");
for (int i = 0; i < 300; i++) {
    for (int j = 0; j < 279; j++) {
        if (audioPwr[i][j] != activitytopowerP[i][j]) {
            printf("\n in C: %f and in P: %f ", audioPwr[i][j],activitytopowerP[i][j]);

        }
    }
    printf("\n");
}
```

Figure 14. comparing the location of both files test

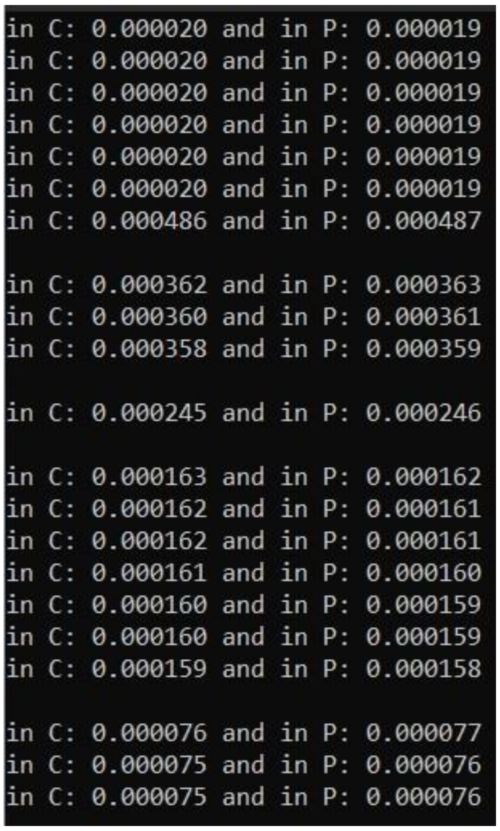


```
in C: 0.001888 & in P: 0.001887
in C: 0.001888 & in P: 0.001887
in C: 0.001888 & in P: 0.001887


in C: 0.002801 & in P: 0.002802










in C: 0.001413 & in P: 0.001412
```

Figure 15. the Output shows the different elements between both files of ElFieldToActivity function



```
D:\Study\S2-2022\ELEC4250\submit_proje

in C: 0.000342 and in P: 0.000341

in C: 0.000426 and in P: 0.000425
in C: 0.000436 and in P: 0.000435
in C: 0.000435 and in P: 0.000434
in C: 0.000433 and in P: 0.000432
in C: 0.000432 and in P: 0.000431
in C: 0.000429 and in P: 0.000428
in C: 0.000428 and in P: 0.000427
in C: 0.000426 and in P: 0.000425

in C: 0.000051 and in P: 0.000050
in C: 0.000050 and in P: 0.000049
in C: 0.000049 and in P: 0.000048
in C: 0.000051 and in P: 0.000050
in C: 0.000050 and in P: 0.000049
in C: 0.000049 and in P: 0.000048
in C: 0.000051 and in P: 0.000050
in C: 0.000050 and in P: 0.000049
in C: 0.000049 and in P: 0.000048
in C: 0.000522 and in P: 0.000523
in C: 0.000521 and in P: 0.000522
in C: 0.000520 and in P: 0.000521
in C: 0.000519 and in P: 0.000520
```



```
in C: 0.000020 and in P: 0.000019
in C: 0.000020 and in P: 0.000019
in C: 0.000020 and in P: 0.000019
in C: 0.000020 and in P: 0.000019
in C: 0.000020 and in P: 0.000019
in C: 0.000020 and in P: 0.000019
in C: 0.000486 and in P: 0.000487

in C: 0.000362 and in P: 0.000363
in C: 0.000360 and in P: 0.000361
in C: 0.000358 and in P: 0.000359

in C: 0.000245 and in P: 0.000246

in C: 0.000163 and in P: 0.000162
in C: 0.000162 and in P: 0.000161
in C: 0.000162 and in P: 0.000161
in C: 0.000161 and in P: 0.000160
in C: 0.000160 and in P: 0.000159
in C: 0.000160 and in P: 0.000159
in C: 0.000159 and in P: 0.000158

in C: 0.000076 and in P: 0.000077
in C: 0.000075 and in P: 0.000076
in C: 0.000075 and in P: 0.000076
```

Figure 16. the output shows the different elements between both files of the ActivitytoPower function

As can be seen from the above figures is that some different values justify why there is a difference in the total sum.