Karin Metzgar
ASTE 546
HW2

Part 1: Data Types

a) I think this is what my issue was with my HW1 Code (I will go back to confirm). But when integer division is performed there is no modulus (or it gets dropped). So 1/20 will result in 0 as that is the integer portion of the answer. And the same for the 10/2 results.
b) >
c) >

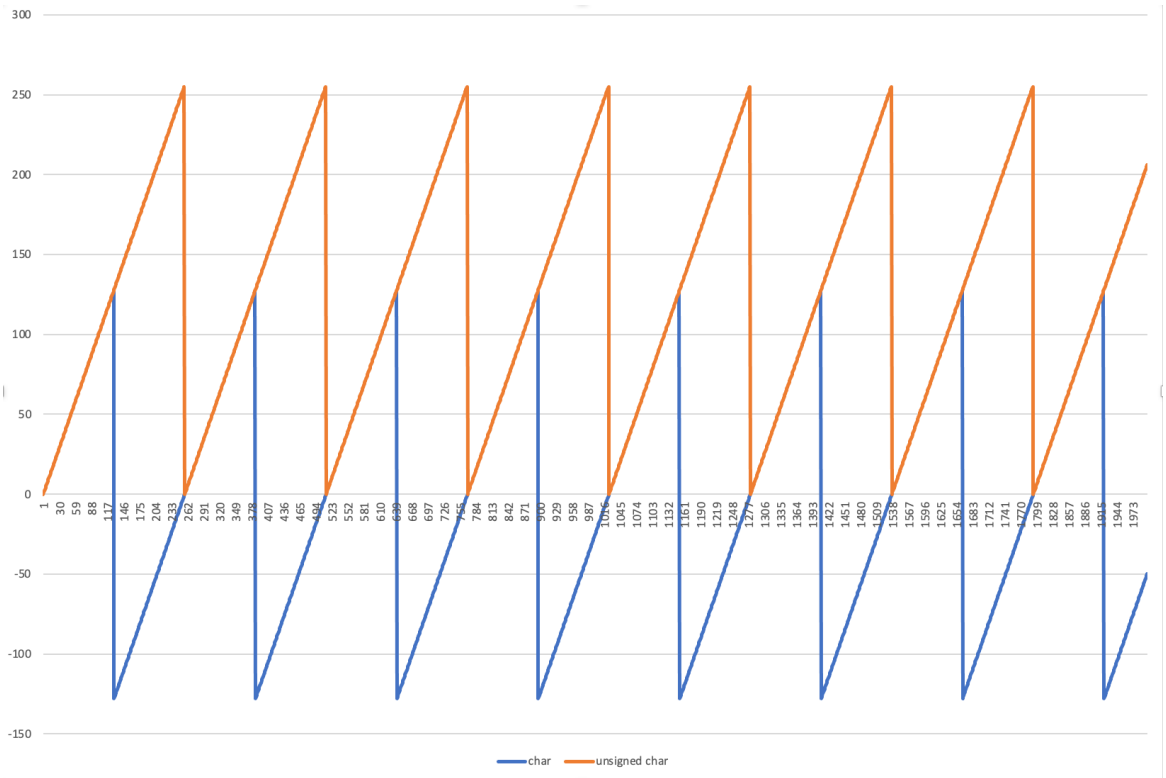
hw2Part1.cpp

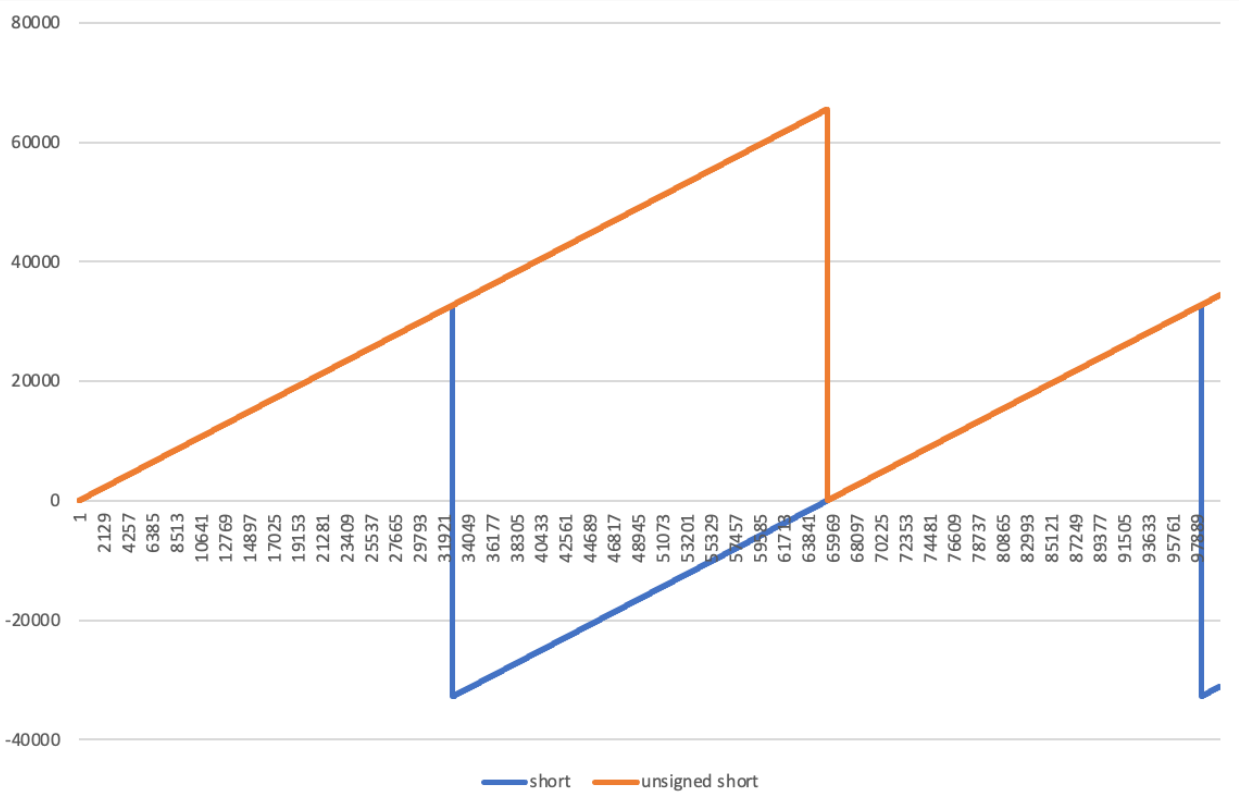
hw2Part1.cpp > main()

```cpp

#include <vector>
#include <iostream>
#include <fstream>
using namespace std;

int main() {
    //a)
    float f = 10/4;
    cout << f << endl;

    //output = 2

    float g = 10/4.0;
    cout << g <<endl;

    //output = 2.5

    float h = 10./4;
    cout << h << endl;

    //output = 2.5

    float i = (float)10/4;
    cout << i << endl;

    //output = 2.5

    int ni = 21;
    cout <<1/(ni-1)<<endl;
    cout<<1/(ni-1.0)<<endl;

    //output = 0
    //output = 0.05

    cout <<10%4<<endl;

    //output = 2

```

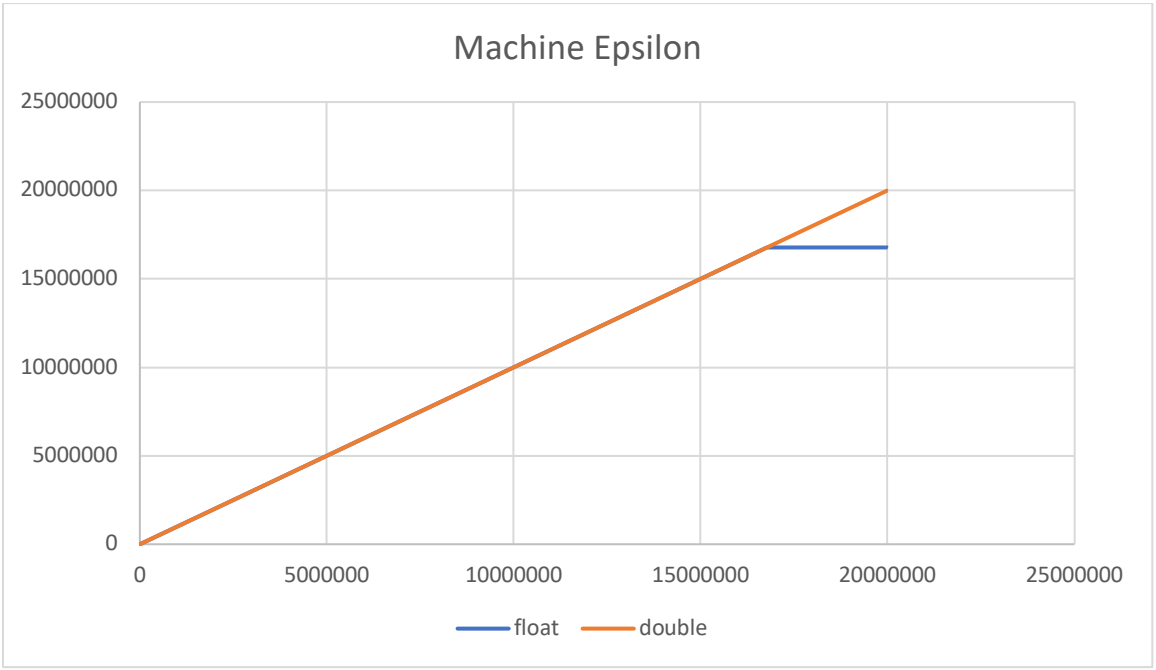
d) Overflow:

```
int main (){

    ofstream out("HW2Part1Overflow.csv");
    out<<"i, char value, unsigned char, short, unsigned short"<<endl;

    //char
    char c;
    unsigned char uc;
    short s;
    uc =0;
    c = 0;
    s=0;
    unsigned short us;

    for (int i=0; i<100000; i++) {
        c = i;
        uc = i;
        s =i;
        us =i;
        out<< i <<"," << (int)c << "," << (int)uc << "," << s <<"," << us << endl;
    }
```





e) It hits the value 1.68E07 at around I = 16750000 and doesn't continue to go up from there



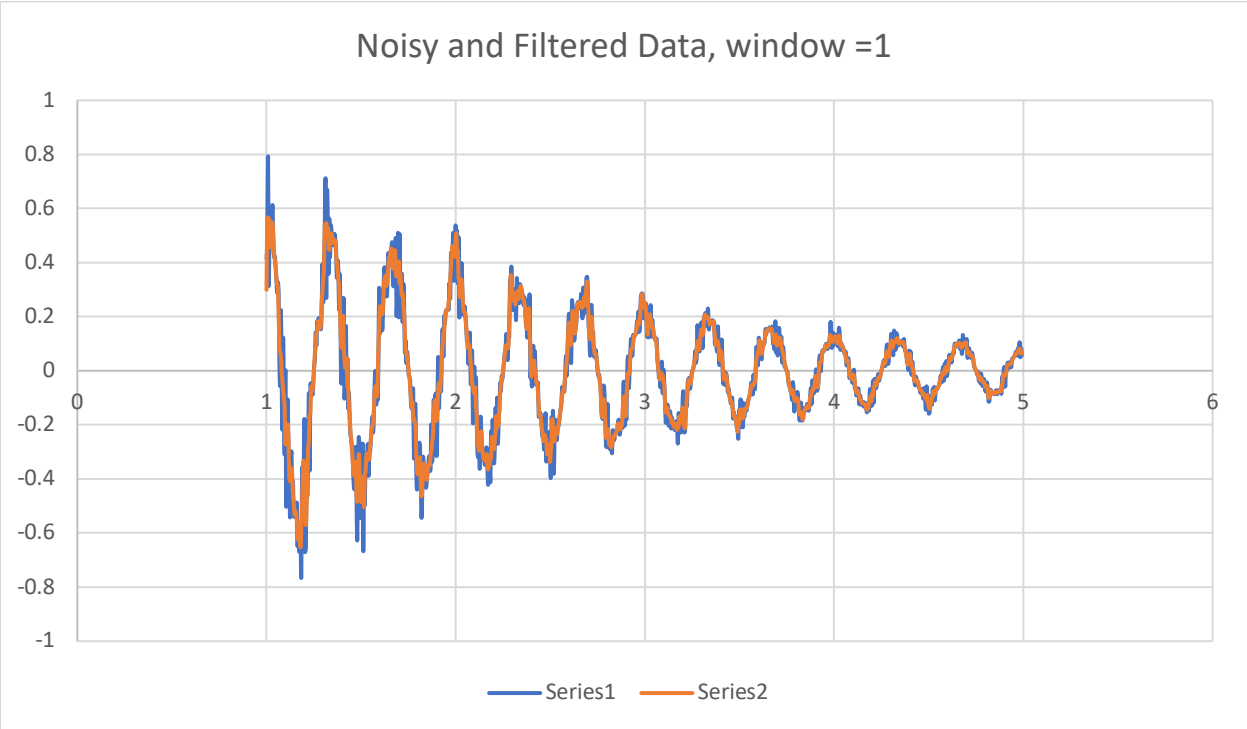
f) 501x501x501 nodes, 20 particles per cell, each grid node store potential, charge density, and 3 components of EF. For each particle 3 position values 3 velocity values:

125,000,000 cells > 2500000000 particles > 15000000000 values total (for particles)
125751501 > 5 values per node = 628757505 values for nodes
= 15628757505 total values. If each value is a double which is 64 bits or 6 bytes each.. that would require 125,030,060,040 bytes = 116 GB

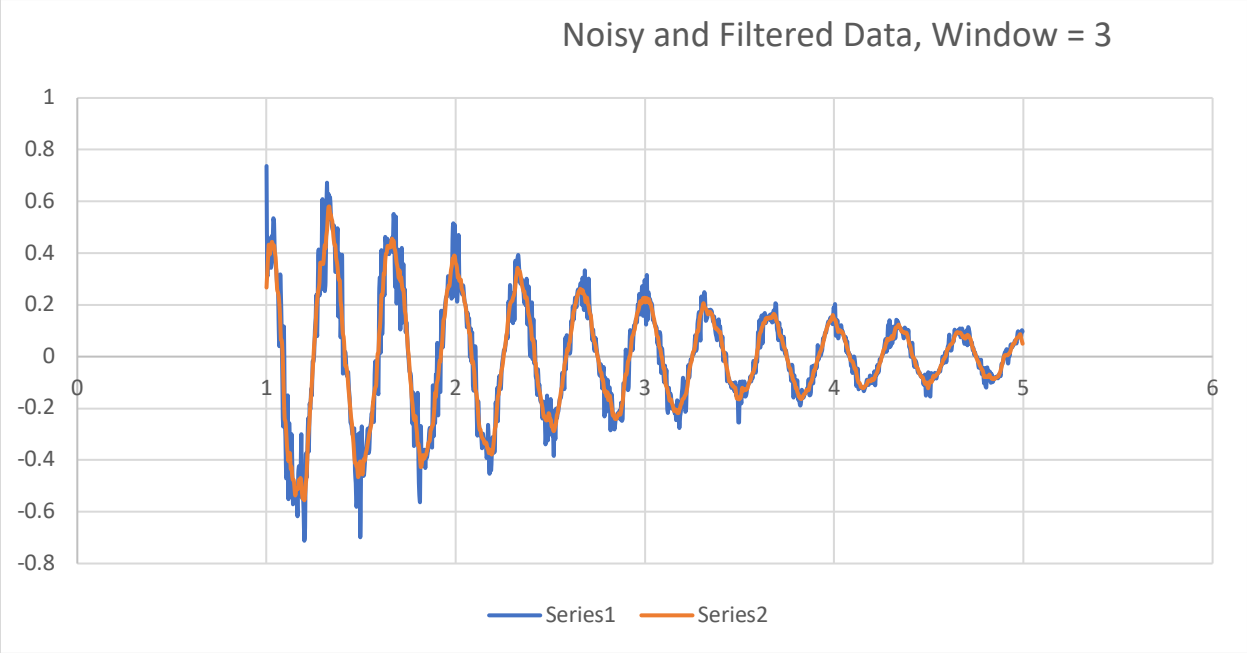
If that is RAM.. then my measly 32GB would not suffice. However, I could spend 260$ on four 32GB Ram cards and get that (assuming my math is right). Although I'd probably still have to close my 89 Chrome tabs…

Part 2: Signal Processor

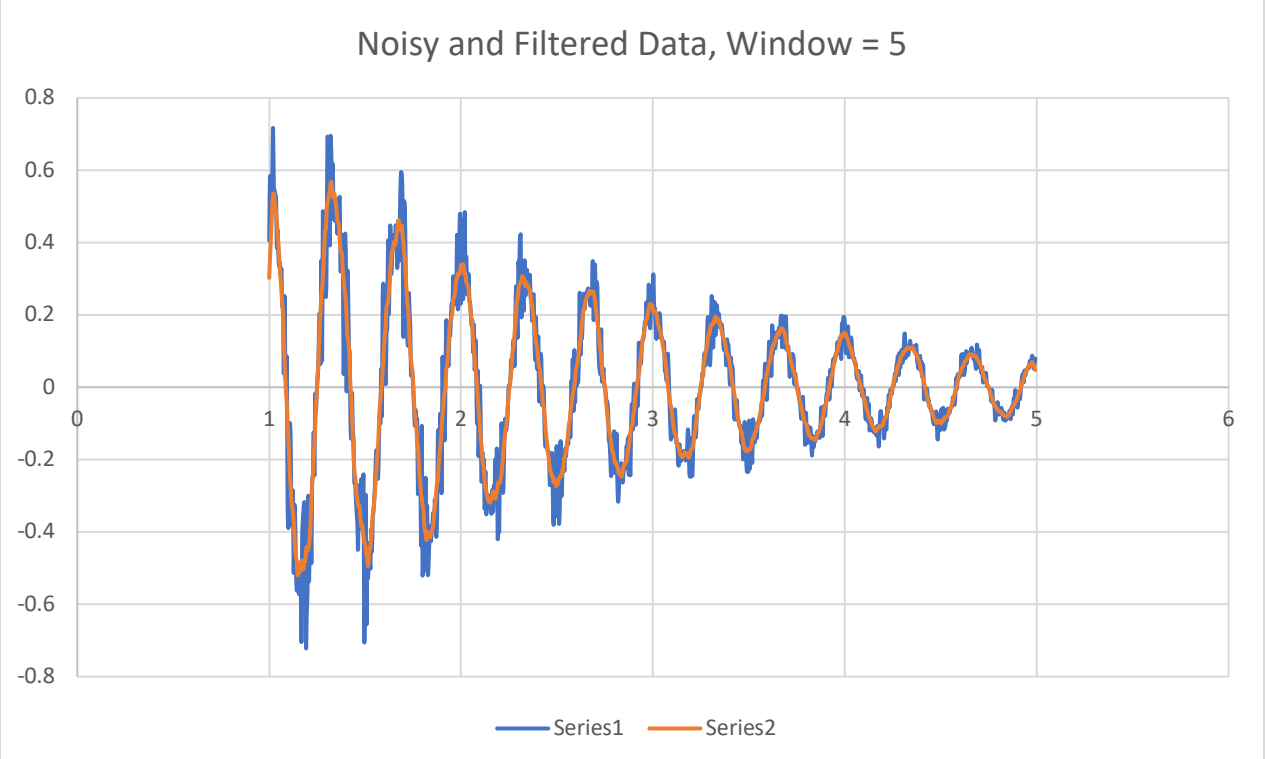
Window = 1



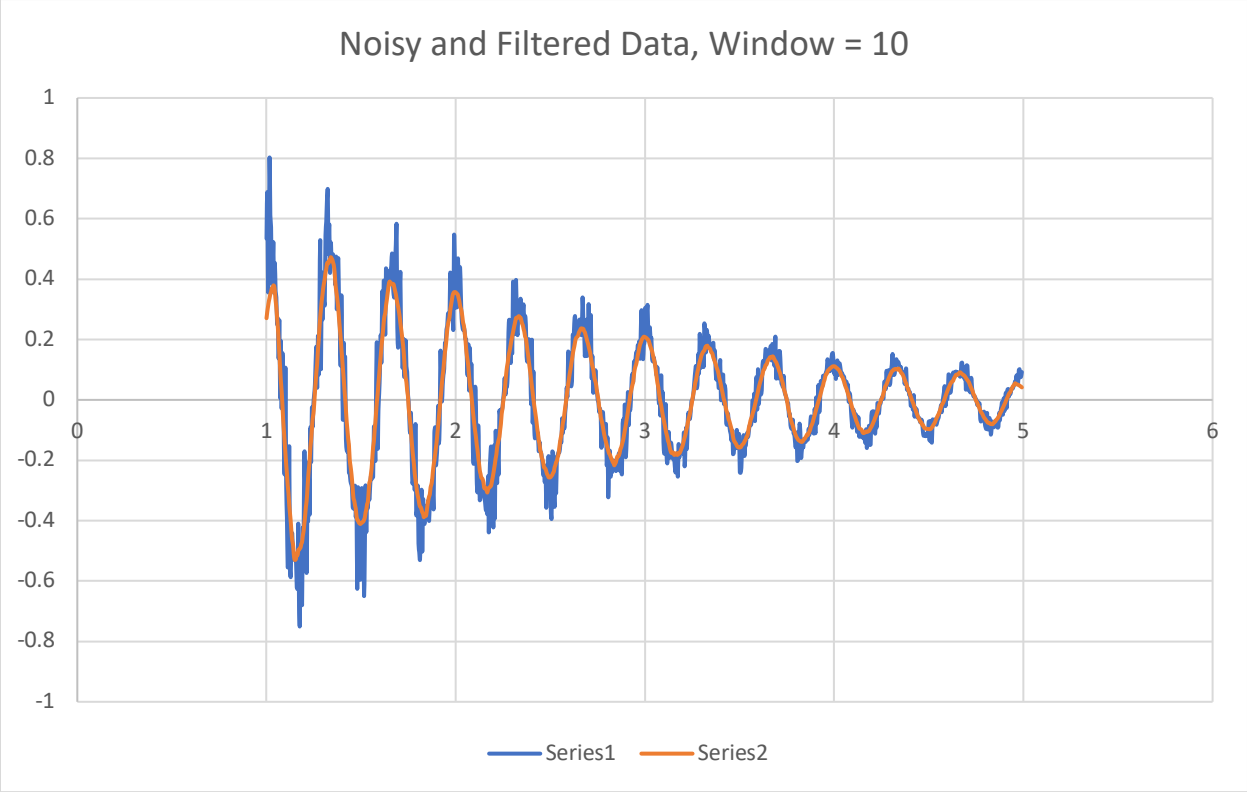
Window = 3



Window = 5



Window = 10


Noisy and Filtered Data, Window = 10
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
0
1
2
3
4
5
6
Series1
Series2

Code:

I defined a function for the equation (the noisy one), a function that read in the noisy data and filtered it outputting an array of filtered data (of the same size), and I defined a function that did the sum of the points within the window.
I created both for vectors and for arrays to try and understand the differences between a reference and a pointer / array vs. vector.

Main ()

```cpp
int main() {
    //print random number
    //cout << rnd()<<endl;

    ofstream out("HW2Part2_SignalProcessor_window3.csv");
    out << "t, ft, filtered ft Array, filtered ft Vector" << endl;

    double ft;
    float t0 =1;
    float tf =5;
    float t;
    float dt;
    int steps = 1001;
    double results[steps];
    double* filteredSignalArray;
    vector<double> filteredSignalVector;
    int window = 3;

    //ran into some more integer division! oops
    dt = ((tf-t0)/static_cast<float>(steps));

    //create noisy signal function
    for(int i=0; i<steps; i++){
        Rnd random;
        float r = random();
        t = t0 + (i*dt);
        ft = mySignalFunction(t,r, window);
        results[i] = ft;
        //out << t << "," << ft << endl;
    }

    //create filtered signal function
    filteredSignalArray = windowFilterPointer(results, steps, window);
    //this translates the results array into a vector, in the below code results is a pointer to the beginning of results
    //and the + steps tells it how many spaces to move past the initial location
    vector<double> vec(results, results + steps);
    filteredSignalVector = windowFilterReference(vec, window);


    //print results
    for ( int j = 0; j <= steps; j++){
        t = t0 + (j*dt);
        out << t << "," << results[j] << ",";
        out << filteredSignalArray[j] << "," << filteredSignalVector[j] << endl;
    }
```

Filter functions

```cpp
double* windowFilterPointer(double *arr, int size, int window)
{
    //this function passes in a pointer to an array, filters the array and outputs the filtered array of the same size
    //this is more light weight and provides more control
    //however it is less safe (cause I might mess up), and it has a fixed size and is thus less flexible.

    //define and allocate memory for filtered array;
    double* filteredArray = new double[size];

    for (int i = 0; i < size; i++){
        filteredArray[i] = sumInRangeArray(arr, size, i, window) / (window * 2 + 1 );
    }

    return filteredArray;
}

vector<double> windowFilterReference(vector<double>& vec, int window){
    //this function passes in the array as a variable size vector
    //using the vector function provides dynamic sizing and bounds checking
    //which are nice because I'm not that good at C++ and might mess something up :)
    //however it is less efficient :(

    //defines the beginning of a new vector but it's empty, we will iterate adding new points to the end
    //thus it is dynamically defined
    vector<double> filteredVector;

    for (auto it = vec.begin(); it != vec.end(); it++){

        int i = distance(vec.begin(), it);
        //push_back adds a new value to the end of vector (or adds a new value)
        filteredVector.push_back( sumInRangeVector(vec,i,window) / (window *2 + 1));
    }

    return filteredVector;
}
```

sumInRange functions

```cpp
double sumInRangeArray(const double* arr, int size, int i,int window){
    //i is the index of interest in the input array
    //windowSize is the size of the window of the filter
    //arr is the array
    //size is size of the array

    double sum = 0;

    int startIndex = max(0,i-window);
    int endIndex = min(size - 1, i + window);

    for(int j = startIndex; j <= endIndex; j++){
        sum += arr[j];
    }

    return sum;
}

double sumInRangeVector(vector<double>& vec, int i, int window){

    double sum = 0;
    int startIndex = max(0,i-window);
    int endIndex = min(static_cast<int>(vec.size()) - 1, i + window);

    for (int j = startIndex; j <= endIndex; j++){
        sum += vec[j];
    }

    return sum;
}
```