Wave\_simulation2 plotting code:

The simulation aims to develop an understanding of the role of spatial coherence in an interferometer. It can be used to efficiently model a wide variety of interferometers made with gratings.

The model incorporate three gratings each located an arbitrary distance away from each other and the source.

The following code plots a graph of the array izx, which shows the behavior of the waves of a beam of electrons before and after each grating in the interferometer.

BEFORE MAIN

First, I included all the required libraries and headers:

#include <stdio.h>

#include <stdlib.h>

#include <math.h>

#include <complex.h>

#include <string>

#include "complex.h"

#include "TCanvas.h"

#include "TGraph.h"

#include "TH2D.h"

#include "TApplication.h"

#include "TROOT.h"

#include "TStyle.h"

#include "SimplePlot.hh"

Then, I defined all the global variables and prototype functions:

//global variables:

double e\_charge = 0.00000000000000000016021765;

double e\_mass = 0.00000000000000000000000000000091093819;

double Plancks = 0.0000000000000000000000000000000006626068;

double Coulomb = 0.00000000898755179;

double pi = 3.14159265358979;

double const\_e = 2.71828182845905;

double chargeratio =0;//strength of image charge(units of e (?))

double eta1 = .4;//G1 open fraction

double eta2 = .4;//G2 open fraction

double d = 0.0000001;// period of grating

double r0 = -4.04;//initial radius of wavefront curvature

double el0 = 0.000001;// initial coherence width

double w0 = 0.00003;// initial beam width

double G1\_z = 0.000001;

double G2\_z = 1;

double G2\_x = 0.00000005; //d/2;

double theta = 0;

double width = 0.00000004;

double thick = 0.000000014;

double wedgeangle = 0;

double tilt =0;

double res = 1000;

double zstart = -0.1;

double zend = 2.1;

double xstart = -0.00020;

double xend = 0.00020;

double ystart = -0.00011;

double yend = 0.00011;

double xpnts = 300;

double ypnts = 300;

double zpnts = 300;

int rows = 300;// rows of ix array

int col = 2;// colomns of ix array

int rowsT =41;// rows of ReT and Imt arrays

double (\*q)[2];//pointer used to modify ix array,for the first time, by gp0, gp1 or gp2 function

double (\*q1)[2];//pointer used to modify ix array, for the second time, by ixgenerator function.

double (\*q2)[2];// pointer used to define the arrays ReT and ImT in the functions gp1 and gp2.

double max;

//Prototype functions:

double zp(double z, double v);//prototype

double w(double z,double r0, double el0, double w0, double energy);//prototype

double el(double z,double r0, double el0, double w0, double energy);//prototype

double v(double z,double r0, double el0, double w0, double energy);//prototype

double sinc(double x);//prototype

double (\*ReTgenerator(double ReT[], double energy))[2];//prototype

double (\*ImTgenerator(double ReT[], double energy))[2];//prototype

double maximumvalue(double arr[][2]);//prototype

double (\*ixgenerator(double a[][2]))[2];//prototype

double (\*gp0(double z,double r0,double el0, double w0, double a[][2],double energy, int rows, int col))[2];//prototype

double (\*gp1(double z12,double r0,double el0, double w0, double a[][2],double energy, int rows, int col))[2];//prototype

double (\*gp2(double z12,double z23, double mytheta, double el1x, double w1x, double r1x, double el1y, double w1y, double r1y, double G2\_x, double a[][2],double

MAIN

{

In the main code, I initialized and defined some other variables which require formulas or functions to be defined:

int main()

{

double energy = ((1.5\*pow(10,-18))/(pow(0.00000000001,2)))\*(1);//((1.5\*pow(10,-18))/(pow(lambda,2)))\*(1); //15000// energy defines lambda and vel, which defnine ix. energy is an argument of most of the functions related to ix.

double zres = (zend-zstart)/zpnts;

double w1=w(G1\_z,r0,el0,w0,energy);

double r1 = v(G1\_z,r0,el0,w0,energy);

double el1=el(G1\_z,r0,el0,w0,energy);

I also initialized the array izx:

double izx[90000];

Then. I started a for loop which goes from i =0 through I =299( 300 values). Each value of i generates an ix array with 300 values each. After being modified, the ix is going to be transmitted to izx array, so at the end of the loop (i=299), there is going to be 90000 (300\*300) values transmitted to izx array:

for (int i=0; i<zpnts; i++)

{

inside the loop I defined zloc, which is the value of z in a determined i. I also initialized the array ix. I had to initialize it here because it avoids garbage values in the array after each loop.

double ix[300][2]={{0}};

double zloc = zstart + i\*zres;

So, I could create the conditions that define the behavior of the wave before and after each grating. I used a pointer (q) in order to return the array ix from the functions gp0, gp1 and gp2. Each function modifies the values of ix (more specifically ix[i][1]) considering its location through z, before or after a grating.

I also used the pointer q1 to modify ix with the ixgenerator function. In this function, each value of ix[i][1] is divided by the maximum value of the whole array:

if (zloc > G2\_z)

{

q = gp2(G2\_z-G1\_z,zloc-G2\_z,theta,el1,w1,r1,el1,w1,r1,G2\_x,ix,energy,rows,col);

q1 = ixgenerator(ix);

}

else

{

if (zloc > G1\_z)

{

q = gp1(zloc - G1\_z, r1, el1, w1,ix,energy,rows,col);

q1 = ixgenerator(ix);

}

else

{

q = gp0(zloc, r0, el0, w0,ix,energy,rows,col);

q1 = ixgenerator(ix);

}

}

Then, I created another loop that helps to define the values of f(from 0 through 90000), which goes through all the length of izx:

for (int j=0; j<zpnts; j++)

{

int f = rows\*i+j;//300\*i+j

Moreover, I defined each value of izx with its respective ix value:

izx[f] = ix[j][1];

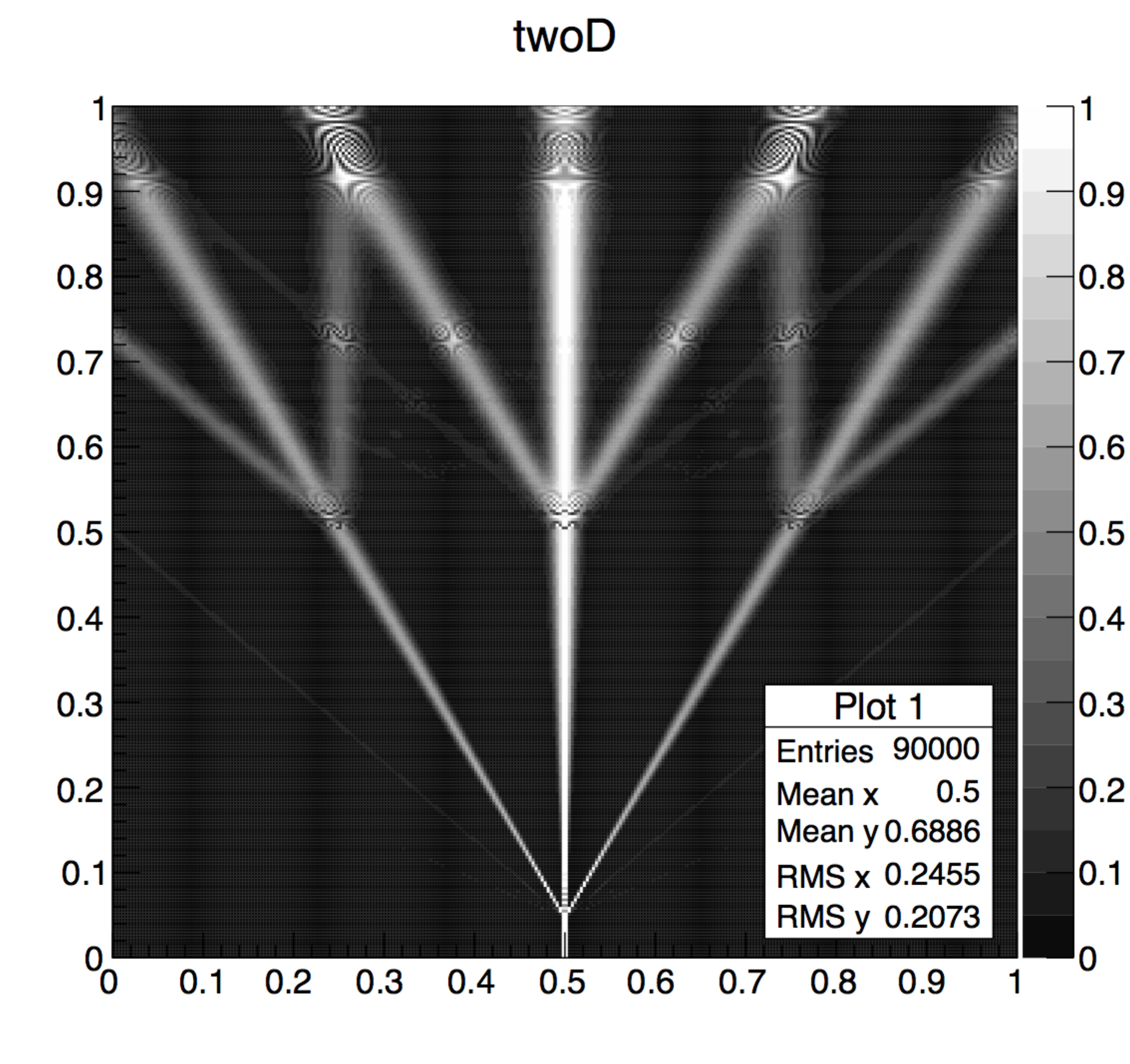
printf("the value of izx is:%0.15f \t %d \t %f \n", izx[f],f, max);

then I closed the two for loops, used the SimplePlot function and closed the main block

SimplePlot::twoD("twoD",izx,0.0,1.0,0.0,1.0,rows,rows);

}

The graph plotted is the following:



The **maximumvalue** is a function that gets the maximum value of each array ix.

First I defined m (maximum value) to 0, then I created a “for loop” which goes through all the values of ix. Then, the condition equals m to the ix value if the ix value is greater than m.

double maximumvalue(double arr[][2]){

double m =0;

for (int i = 0; i < rows; i++)

{

if (m<arr[i][1])

{

m = arr[i][1];// finds the maximum value of the array

}

}

return m;

//printf("the value of m is : %f\n",m);

}

The **ixgenerator** is a function that modifies all the ix[i][1] values. With the max value provided by the maximumvalue function, it divides each ix[i][1] value by the maximum value of the respective ix array.

I used the pointer q to relate ix to this function.

double (\*ixgenerator(double a[][2]))[2]

{

double max = maximumvalue(a);

for (int i=0; i<rows; i++)

{

if (a[i][1]<0)

{

a[i][1]=0;

}

a[i][1] = a[i][1]/max;//divides each element of the array by max

}

return a;

}

The **\*ReTgenerator** is a function that defines all the elements of the array ReT:

First I initialize all the required local variables:

(if I initialize fc, ph and ex as double, the calculation returns a wrong value)

double eta = width/d;

double vel = pow(2\*energy\*e\_charge/e\_mass,1/2);

double alpha = wedgeangle\*pi/180;

double beta = tilt\*pi;

double xmin;

double xmax;

float fc;

float ph;

float ex;

int j;

Then I created if/else conditions that defines the values of xmax and xmin. Those values depend on the local variable vel, which is defined by the argument variable energy:

if (beta>=0){

xmin= width\*(1/res - cos(beta)/2);

if (beta<=alpha) {

xmax=(width\*cos(beta))/2-width/res;

}

else

{

xmax=width\*cos(beta)/2-width/res+thick\*(tan(alpha)-tan(beta));

}

}

else

{

xmax = (width\*cos(beta)/2)-width/res;

if (fabsl(beta)<=alpha) {

xmin = -((width\*cos(beta))/2)+width/res;

}

else

{

xmin = -((width\*cos(beta))/2)+width/res - thick\*(tan(alpha)-tan(beta));

}

}

Moreover, I made a nested for loop, which defines the local variables fc and pc, that, in the same nested loop, are used to define ReT:

for(int n=-((rowsT-1)/2);n<=((rowsT-1)/2);n++)//

{

for(ex=xmin; ex<xmax; ex+=width/res)//copied from above

{

fc = 2\*pi\*n\*ex/d;

ph = -width\*thick\*chargeratio\*pow(e\_charge,2)\*(2\*pi\*Coulomb/Plancks)/(vel\*(.25\*pow(width,2)-pow(ex,2)));

j=n+((rowsT-1)/2);

ReT[j] += cos(ph+fc);

}

}

For last, I divided all the values of ReT by the global variable res:

for (int i=0; i<rowsT; i++) {

ReT[i] = ReT[i]/res;

}

}

The **ImTgenerator** function is almost the same as ReTgenerator, the only difference is that at the end, instead of defining ImT with ReT[j] += cos(ph+fc);, it defines ImT with ImT[j] += sin(ph+fc);

**gp0** is a function the defines the behavior of the wave before the first grating. . It returns a pointer to an array of 2 columns.

First, I initialized w1, which is a function of z(changes with z and energy) and used the function w to defined it. Then, I created a for loop which goes through all the length of a(ix). Inside this loop are the formulas that defines each column of a(ix):

double (\*gp0(double z,double r0,double el0, double w0, double a[][2],double energy, int rows, int col))[2]

{

double jj;

double w1 = w(z,r0,el0,w0,energy);

for(int i=0; i<rows; i++)

{

a[i][0]= xstart+(i)\*((xend-xstart)/(xpnts-1));

jj =pow((a[i][0]/w1),2);// I had to break it appart to make it work

a[i][1]=exp(-(pi\*jj));

}

return a;

}

GP1

**The gp1** function defines the behavior of the wave between the first and second gratings. It returns a pointer to an array of 2 columns.

Its arguments are the following:

gp1(double z12,double r1,double el1, double w1, double a[][2], double energy, int rows, int col)

First I initialized all the required variables:

double coef;

double cutoff=pow(10,-3);

double lim=5;

double lambda = sqrt((1.5\*pow(10,-18))/(energy));

double eta = width/d;

double vel = pow(2\*energy\*e\_charge/e\_mass,1/2);

double alpha = wedgeangle\*pi/180;

double beta = tilt\*pi;

double w2=w(z12,r1,el1,w1,energy);

double r2 = v(z12,r1,el1,w1,energy);

double el2 = el(z12, r1, el1, w1,energy);

Then, I initialized and defined pos, ReT and ImT. The arrays ReT and ImT were defined with the functions ReTgenerator and ImTgenerator and the pointer q3:

int pos[41]={0};

for (int i=0; i<rowsT; i++)

{

pos[i]=i-((rowsT-1)/2);

}

double ReT[41]{0};

q2 = ReTgenerator(ReT,energy);

double ImT[41]={0};

q2 = ImTgenerator(ImT,energy);

Then, I defined the values of the first column of a(ix) (argument of this function):

for (int i=0; i<rows; i++) {

a[i][0]= xstart+(i)\*((xend-xstart)/(xpnts-1));

}

So, I made three loops, i,n and m. i goes through all the length of a(ix), and n and m are related to the formulas that defines the second columns of a(ix). These parameter are related to the values of each horizontal “line” of the final plot:

for (int i=0; i<rows; i++) {

for (int n=-lim; n<=lim; n++) {

for (int m=-lim; m<=lim; m++) {

Inside the loop I defined the variables dn, dm and test1.

dn and dm are related with n and m and therefore, with the formulas that defines the second column of a(ix):

double dn =n-m;

double dm = (m+n)/2;

double test1=0;

I created test1 to use in the first condition of the loop. By initializing it to 0, the condition will be false for “if (test1==1)”, and then the loop will execute the else block:

if (test1==1)

{

coef = sinc(eta1\*pi\*n)\*(sinc(eta1\*pi\*m)\*pow((eta1), 2));

}

else

{

coef = ReT[n + ((rowsT-1)/2)]\*ReT[m + ((rowsT-1)/2)]+ImT[n + ((rowsT-1)/2)]\*ImT[(m + ((rowsT-1)/2))];//n+(rowsT-1)/2 represents the position of the element n in the array ReT. if rowsT is 41 and n is -5 for example, the position would be 15.

}

coef = coef\*exp(-pi\*(dn\*(lambda\*z12/(pow(d\*el2,2)))));

if (isfinite(coef)==0)// added isfinite macro in order to avoid inf values

{

coef=0;

}

Coef is a coefficient that is also used to define the second column of a(ix). I had to use this condition:

if (isfinite(coef)==0)

{

coef=0;

}

because some of the values of coef were nan.

Then, the second condition is if coef is greater than cutoff (0.001), the loop defines the second column of ix by the formula:

if (coef>=cutoff)

{

a[i][1] = a[i][1] + (coef\*exp(-pi\*pow(((a[i][0]-dm\*lambda\*z12/d)/w2),2)\*cos(2\*pi\*(dn/d)\*(a[i][0]-dm\*lambda\*z12/d)\*(1-z12/r2))));

continue;

}

}

}

}

Then, I closed the loop and returned a (pointer to a) which defines ix in the main block:

return a;

}

Gp2

The **gp2** function defines the behavior of the wave after the second grating. It returns a pointer to an array of 2 columns.

Its arguments are the following:

double z12,double z23, double mytheta, double el1x, double w1x, double r1x, double el1y, double w1y, double r1y, double G2\_x, double a[][2], double energy, int rows, int col)

First I initialized all the required variables:

double lambda = sqrt((1.5\*pow(10,-18))/(energy));

double res = 1000;

double eta = width/d;

double vel = pow(2\*energy\*e\_charge/e\_mass,1/2);

double alpha = wedgeangle\*pi/180;

double beta = tilt\*pi;

double theta = pi\*mytheta/180;

double d1=d;

double d2=d;

double z13 = z12+z23;

double phi =0;

double cutoff = 0.001;

double lim =5;

double \_Complex coef;

double dn = 0;

double dm =0;

double m=0;

double n=0;

int a5 =0;

int b =0;

int c5 =0;

int d5=0;

double test1=0;

double el3x = el(z13, r1x, el1x, w1x, energy);//G2z - G1z + zstart + 0\*zres, r1, el1, w1

double w3x = w(z13,r1x,el1x,w1x, energy);

double v3x = v(z13,r1x,el1x,w1x, energy);

double el3y = el(z13,r1y,el1y, w1y, energy);

double w3y = w(z13,r1y,el1y,w1y, energy);

double v3y = v(z13,r1y,el1y,w1y, energy);

Then, I initialized and defined pos, ReT and ImT. The arrays ReT and ImT were defined with the functions ReTgenerator and ImTgenerator and the pointer q3:

int pos[41]={0};

for (int i=0; i<rowsT; i++)

{

pos[i]=i-((rowsT-1)/2);

}

double ReT[41]={0};

q2 = ReTgenerator(ReT,energy);

double ImT[41]={0};

q2 = ImTgenerator(ImT,energy);

I also initialized and defined the first column of the array phix, and I defined the first column of the array a(argument of this function):

for (int i=0; i<rows; i++) {

a[i][0]= xstart+(i)\*((xend-xstart)/(xpnts-1));

}

double phix[300][2]={0};

for (int i=0; i<rows; i++) {

phix[i][0]= xstart+(i)\*((xend-xstart)/(xpnts-1));

}

So, I started five loops, i, m1,m2,n1 and n2. i is related with the length of the a(ix) array, and the other are related with the formulas that with generate the second column of a(ix). These parameter are related to the values of each horizontal “line” of the final plot:

for (int i=0; i<rows; i++){

for (int m1=-lim; m1<=lim; m1++) {

for (int m2=-lim; m2<=lim; m2++) {

for (int n1=-lim; n1<=lim; n1++) {

for (int n2=-lim; n2<=lim; n2++) {

Inside the loop, I defined the variables, dn,n,dm,m, which are related with the loop, and a5,b,c5,d5 which are related with the index of the arrays ReT and ImT.(for example, if m1=-5 and ReT is an array with 41 values, a5 would be 15, which would be the index in ReT equivalent to the index -5 from m1):

dn =n1-n2;

n = ((double)(n1+n2))/2;

dm = m1-m2;

m = ((double)(m1+m2))/2;

a5 = m1 + ((rowsT-1)/2);

b = m2 + ((rowsT-1)/2);

c5 = n1 + ((rowsT-1)/2);

d5 = n2 + ((rowsT-1)/2);

As in gp1 function, I also create the variable test1 to use in the first condition of the loop. By initializing it to 0, the condition will be false for “if (test1==1)”, and then the loop will execute the else block:

if (test1==1)

{

coef = sinc(eta1\*pi\*m1)+ 0\*\_Complex\_I;

coef = coef\*(sinc(eta1\*pi\*m2+ 0\*\_Complex\_I));

}

else

{

coef = ReT[a5][1]+ImT[a5][1]\*\_Complex\_I;

coef = coef\*((ReT[b][1]-ImT[b][1]\*\_Complex\_I));

}

coef = coef\*(ReT[c5][1] + ImT[c5][1]\*\_Complex\_I);

coef = coef\*(ReT[d5][1] + ImT[d5][1]\*\_Complex\_I);

coef=coef\*(exp(-pi\*pow(((dn\*sin(theta)\*lambda\*(z23))/(d2\*el3y)),2)));

coef=coef\*(exp(-pi\*pow((lambda\*z23\*(dn\*cos(theta)+dm\*z13/z23)/(d1\*el3x)),2)));

The second condition is if the real or imaginary components of coef are greater than cutoff(0.001). When this is true, a variable called phi is calculated and then used to define the second columns of phix.

Phix is then used to define the elements of the second column of a (ix):

if (((\_\_real\_\_ coef)>=cutoff) || ((\_\_imag\_\_ coef)>=cutoff)) {

phi = dn\*n\*(1-z23/v3x)\*pow((cos(theta)),2) + dn\*n\*(1-z23/v3y)\*pow((sin(theta)),2) + dn\*m\*(1-z13/v3x)\*cos(theta);

phi = phi +(dm\*n\*(1-z13/v3x)\*cos(theta) + dm\*m\*(z13/z23)\*(1-z13/v3x));

phi = phi\*(2\*pi\*lambda\*z23/(pow(d1,2)));

phi = phi - (2\*pi\*dn\*G2\_x/d2);

phix[i][1] = ((phi-(2\*pi\*(phix[i][0])/d2)\*(dn\*cos(theta)\*(1-z23/v3x) + dm\*(1-z13/v3x))));

a[i][1] = a[i][1] + ((((\_\_real\_\_ coef)\*cos(phix[i][1]) - (\_\_imag\_\_ coef)\*sin(phix[i][1]))\*exp(-pi\*pow(((phix[i][0]-(lambda\*z23/d1)\*(n\*cos(theta)+m\*(z13/z23)))/w3x),2))));

Then, I closed all the loops and returned the array a as a pointer.

}

}

}

}

}

return a;

}