Wave\_simulation1 plotting code:

This code plots three graphs of the array ix, which shows the behavior of the waves. The graph made by the gp0 functions shows the behavior of the waves before the first grating. The one made by the gp1 function shows the behavior of the waves between the first and second grating. And the graph made by the gp2 function shows the behavior of the waves after the second grating.

BEFORE MAIN

First, I included all the required libraries and headers:

#include <stdio.h>

#include <stdlib.h>

#include <math.h>

#include <complex.h>

#include <complex>

#include <ccomplex>

#include "complex.h"

#include "TCanvas.h"

#include "TGraph.h"

#include "TApplication.h"

#include "TROOT.h"

#include "SimplePlot.hh"

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

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;

int rowsT = 41;

double (\*q)[2];

//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];

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

void gp0(double z,double r0,double el0, double w0,double energy);

void gp1(double z,double r0,double el0, double w0,double energy);//prototype

void gp2(double z12,double z23, double mytheta, double el1x, double w1x, double r1x, double el1y, double w1y, double r1y, double G2\_x, double energy); // prototype

int x2pnts(double \*arr, int r, int value);//prototype

MAIN

{

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

int main( )

{

double energy = ((1.5\*pow(10,-18))/(pow(0.00000000001,2)))\*(1);

double zres = (zend-zstart)/zpnts;// matches with Mathematica code value

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

double w1=w(G1\_z,r0,el0,w0,energy);//gp1

double r1 = v(G1\_z,r0,el0,w0,energy);//gp1

double el1=el(G1\_z,r0,el0,w0,energy);//gp1

Then, in the end of the main function I used the void functions; gp0, gp1 and gp2. Each of these function plots a graph of the wave behavior in its location. After that, I closed the main function:

gp0( zstart + 1\*zres,r0,el0,w0, energy);

gp1(zstart-G1\_z+zres\*100,r1,el1,w1, energy);

gp2(G2\_z-G1\_z,zstart+0\*zres,theta,el1,w1,r1,el1,w1,r1,G2\_x, energy);

}

The **x2pnts** is a function that finds a value in an array and returns its respective index(ix[i][0])

int x2pnts(int value, int \*arr)

{

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

{

if (value == \*(arr+i))

{

return(i);

}

}

printf("Error! n or m value does not match with any value of x2pnt array\n");

}

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 variable s 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**

**gp0** is a function the defines the behavior of the wave before the first grating. Its arguments are the following:

void gp0(double z,double r0,double el0, double w0, double energy)

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

void gp0(double z,double r0,double el0, double w0, double energy)

{

double jj;

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

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

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

{

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

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

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

printf("the value of ix[i][1] is %f\n",ix[i][1]);

}

In the end of this function I separated the array a into two vector, so that I could use the SimplePlot function.:

double ix1[300]={0};

double ix2[300]={0};

for (int i=0; i<300; i++) {

for (int j=0; j<2; j++) {

if (j==0) {

ix1[i]=ix[i][j];

}

if (j==1) {

ix2[i] = ix[i][j];

}

}

}

SimplePlot::graph("gp1 graph",ix1,ix2,300);

}

**GP1**

**The gp1** function defines the behavior of the wave between the first and second gratings. Its arguments are the following:

void gp1(double z12,double r1,double el1, double w1, double energy)

First I initialized all the required local 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. I defined ReT and ImT with the functions (\*ImTgenerator ) and (\*ReTgenerator )

int pos[41]={0};

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

{

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

}

double ReT[41]{0};

q = ReTgenerator(ReT,energy);

double ImT[41]={0};

q = ImTgenerator(ImT,energy);

(\*ImTgenerator ) Then, I initialized the array ixand defined the values of its first column:

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

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

ix[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 separated the array ix into two vector, in order to plot it by using the function SimplePlot:

double ix1[300]={0};

double ix2[300]={0};

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

for (int j=0; j<2; j++) {

if (j==0) {

ix1[i]=ix[i][j];

}

if (j==1) {

ix2[i] = ix[i][j];

}

}

}

SimplePlot::graph("gp1 graph",ix1,ix2,rows);

}

Gp2

The **gp2** function defines the behavior of the wave after the second gratings. Its arguments are the following:

void gp2(double z12,double z23, double mytheta, double el1x, double w1x, double r1x, double el1y, double w1y, double r1y, double G2\_x, double energy)

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. I defined ReT and ImT with the functions (\*ImTgenerator ) and (\*ReTgenerator ):

int pos[41]={0};

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

{

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

}

double ReT[41]={0};

q = ReTgenerator(ReT,energy);

double ImT[41]={0};

q = ImTgenerator(ImT,energy);

I also initialized and defined the first column of the array phix, and I defined the first column of the array ix:

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

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

ix[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):

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. (to define them I used the function x2pnts):

dn =n1-n2;

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

dm = m1-m2;

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

a5 = (x2pnts(m1, (int \*)pos));

b = (x2pnts(m2, (int \*)pos));

c5 = (x2pnts(n1, (int \*)pos));

d5 = (x2pnts(n2, (int \*)pos));

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))));

ix[i][1] = ix[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 separated the array ix into two vector in order to plot it by using the function SImplePlot:

}

}

}

}

}

}

double ix3[300]={0};

double ix4[300]={0};

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

for (int j=0; j<2; j++) {

if (j==0) {

ix3[i]=ix[i][j];

}

if (j==1) {

ix4[i] = ix[i][j];

}

}

}

SimplePlot::graph("gp2 graph",ix3,ix4,rows);

}