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 lambda = 0.00000000001;

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;// number of rows of main array

int rowsT = 41;// number of rows of ReT and ImT arrays

double (\*q)[2];// pointer used to call the functions (\*ReTgenerator(double ReT[][2]))[2] and gp1(double z,double r0,double el0, double w0)

//prototype functions:

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

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

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

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

double sinc(double x);//prototype

double (\*ReTgenerator(double ReT[][2]))[2];

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

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

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

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

int main( )

{

double lam = pow((150)/(4000),1/2)\*pow(10,-10);// double lam = 1.936E-10//

///////////Fourier Components Variables//////////

double energy = (1.5\*pow(10,-18))/(pow(lambda,2)); //15000// energy

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

double res = 1000;// double res = 1000;

double eta = width/d;

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

double alpha = wedgeangle\*pi/180;

double beta = tilt\*pi;// since tilt=0, beta=0.

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

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

double w1;

double r1;

double el1;

double el2;

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

//printf("the value of w1 is: %.12f\n",w1);// matches with Mathematica value of w1

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

//printf("the value of r1 is: %.12f\n",r1);//matches with Mathematica value of r1

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

//printf("the value of el1 is: %.12f\n",el1); //matches with Mathematica value of el1

el2 = el(zstart-G1\_z+zres\*100, r1, el1, w1);//gp2

//printf("the value of el2 is: %.12f\n",el2);// matches with mathematica value

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);

// gp1(0.00266667,r1,el1,w1);

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

// printf("the value of el2 is: %.12f\n",el2);

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

}

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

I first initialized arr2, which will correspond to each value of the second column of the array.

int x2pnts(float \*arr, int r, int value)// put number of rows and columns as arguments?

{

int arr2 = {0};

int ans;

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

{

arr2 = \*(arr+i\*col);//defines arr2 with each value of the array with the pointer \*arr

if (arr2 == value)// if arr2 equals value, the function will return the respective index (ix[i][0])

{

ans = (\*((arr+(i)\*2) + 1));

return(i);

}

}

}

(\*ReTgenerator is a function that create and define the array ReT.

In this function I first initialize all variables required, then I used a condition and two sub conditions to define the variables xmin and xmax:

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));

}

}

xmax:After, I created two for loop to define the variables fc and ph. One of the for loop conditions depend on the size of the arrays ReT and ImT, which are defined as a global variable of the code.( rowsT). The second loop condition depends on the values of xmin and xmax calculated above:

for(int n=-((rowsT-1)/2);n<=((rowsT-1)/2);n++)//n=((rowsT-1)/2): for example, if rowsT=41, n=20

{

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

{

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)));

Inside the same loops, I used the values of the variables fs and ph to define ReT:

j=n+((rowsT-1)/2);// for example, if rowsT=41, j=n+20;

ReT[j][1] += cos(ph+fc);

}

}

Then, after closing the loops I created another loop to divide each value of ReT by the variable res:

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

ReT[i][1] = ReT[i][1]/res;

}

}

(\*ImTgenerator )is a function that create and define the array ImT. It function is pretty much the same as (\*ReTgenerator) function, the only difference is in the formula to define ImT(ReT). In (\*ReTgenerator) function, the formula was:

ReT[j][1] += cos(ph+fc);

And in (\*ImTgenerator), the formula is:

ImT[j][1] += sin(ph+fc);

**Gp0**

**gp0** is a function the defines the behavior of the wave before the first grating.

First, I initialized w1, which is a function of z(changes with z) 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):

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

{

double w1;

double jj;

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

double a[300][2] = {0};//size of these arrays are related with rows variable. So, if there are any change in rows, their size should also be changed.

double ix1[300]={0};

double ix2[300]={0};

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

{

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

jj =pow((a[i][0]/w1),2);

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

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

}

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

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

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

if (j==0) {

ix1[i]=a[i][j];

}

if (j==1) {

ix2[i] = a[i][j];

}

}

}

SimplePlot::graph("gp0 graph",ix1,ix2,rows);

}

**GP1**

**The gp1** function defines the behavior of the wave between the first and second gratings.

First I initialized all the required variables:

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

double coef;

double cutoff=pow(10,-3);

double lim=5;

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 el2;

double w2;

double r2;

long double xmin;

long double xmax;

float fc;

float ph;

float ex;

int j;

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

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

//printf("the values of r2 and w2 are: %f \t %f \n",r2,w2);

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

//printf("the value of el2 is : %0.12f\n",el2);

Then, I initialized and defined ReT and ImT with the functions (\*ImTgenerator ) and (\*ReTgenerator )

double ReT[41][2]={{0}};

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

ReT[i][0]=i-((rowsT-1)/2);

}

q = ReTgenerator(ReT);

double ImT[41][2]={{0}};

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

ImT[i][0]=i-((rowsT-1)/2);

}

q = ImTgenerator(ImT);

(\*ImTgenerator ) Then, I initialized the array ix(a) and 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));

//printf("the values of ix[i] ddand i are: %f\t and %d\n",ix[i][1],i);//different format, but same values as in the Mathematica code

}

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):

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).

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.

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 should follow to define the second column of ix by the formula:

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))));

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.

First I initialized all the required variables:

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

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 w1;

double r1;

double el1;

double el2;

double theta = pi\*mytheta/180;

double d1=d;

double d2=d;

double z13 = z12+z23;

long double xmin;

long double xmax;

float fc;

float ph;

float ex;

double phi =0;

double cutoff = 0.001;

double lim =5;

double \_Complex coef;

//double complex coef;

double dn = 0;

double dm =0;

double m=0;

double n=0;

int a =0;

int b =0;

int c=0;

int d5=0;

double test1=0;

// printf("the values of theta, d1,d2 and z13 are: %0.15f \t %0.15f \t %0.15f \t %0.15f \n",theta,d1,d2,z13);// the values match with the mathematica values.

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

// printf("the value of el3x is: %0.15f\n",el3x);//value matches with mathematica

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

//printf("the value of w3x is: %0.15f\n",w3x);//value matches with mathematica

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

// printf("the value of v3x is: %0.15f\n",v3x);//value matches with mathematica

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

// printf("the value of el3y is: %0.15f\n",el3y);//value matches with mathematica

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

// printf("the value of w3y is: %0.15f\n",w3y);//value matches with mathematica

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

//printf("the value of v3x is: %f\n",v3x);//value matches with mathematica

Then, I initialized and defined ReT and ImT with the functions (\*ImTgenerator ) and (\*ReTgenerator ):

double ReT[41][2]={{0}};

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

ReT[i][0]=i-((rowsT-1)/2);

//printf("value of ReT is: %f and j is: %d \n",ReT[i][1],(i-20)); ok

}

q = ReTgenerator(ReT);

double ImT[41][2]={{0}};

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

ImT[i][0]=i-((rowsT-1)/2);

//printf("value of ImT is: %f and j is: %d \n",ImT[i][1],(i-20)); ok

}

q = ImTgenerator(ImT);

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));

//printf("the values of ix[i] and i are: %f\t \t %d\n",ix[i][0],i);//different format, but same values as in the Mathematica code

}

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

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

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

//printf("the values of phix[i] and i are: %f\t \t %d\n",phix[i][0],i);//different format, but same values as in the Mathematica code

}

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((float \*)ReT, rowsT, m1));

b = (x2pnts((float \*)ReT, rowsT, m2));

c5 = (x2pnts((float \*)ReT, rowsT, n1));

d5 = (x2pnts((float \*)ReT, rowsT, n2));

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);

}