**M3SC-Scientific Computing**

Exercise 1c

Question1 :

The code that passes the arguments as an array is the following :

Q1main:

#include <stdio.h>

#include <math.h>

#include <float.h>

int lin\_root(double \*,double \*);

int quad\_roots(double \*, double \*);

int rcubic\_roots(double \*,double \*);

int main(void){

printf(" Name: <Alexandre Elkrief> \n CID: <00732974> \n LIBRARY NO: <0246431227> \n Email Address: <ae2212@ic.ac.uk> \n Course Code: <M3SC>\n");

double A,B,C;

double a[3],root[4];

scanf("%lf %lf %lf", &a[2], &a[1], &a[0]);

int rcubic\_case, quad\_case;

rcubic\_case=rcubic\_roots(a,root);

printf("Roots= %lf, %lf, %lf \n",root[1], root[2], root[3]);

}

lin\_root:

#include <stdio.h>

#include <math.h>

#include <float.h>

//lin\_root function solves a linear equation taking care of the cases where a1 or/and a0 =0.

//each case returns a special a value which will then be interpreted by the quad\_roots function

int lin\_root(double \*a,double \*root)

/\* <Elkrief>\_<Alexandre>\_exer\_1c \*/

{if (a[1]!=0){

root[0] = -a[0]/a[1];

return(1);}

else if (a[1]==0 && a[0]==0){

return(0);}

else {

return(-1);}

}

Quad\_roots:

#include <stdio.h>

#include <math.h>

#include <float.h>

int lin\_root(double \*,double \*);

int quad\_roots(double \*a, double \*root)

/\* <Elkrief>\_<Alexandre>\_exer\_1c \*/

{double d, dr, changeorder;

int lin\_case;

//printf("quadratic: x^2+%lf\*x+%lf \n", a[1],a[0]);

//the function calls lin\_root and integrates output cases it.

if (a[2]==0){lin\_case=lin\_root(a,root);

switch(lin\_case){

case 1:{return(-1);

break;}

case 0:{return(-2);

break;}

case -1:{return(-3);

break;}

}

}

if (a[0]==0){

lin\_case=lin\_root(a, root);

//ordering the roots in ascending order

if (root[1]<0){

root[2]=0;

return(2);

}

else {

root[2]=root[1];

root[1]=0;

return(2);

}

}

d=a[1]\*a[1]-4\*a[2]\*a[0];

//printf("d= %lf\n",d);

if (d>0){

/\*condition on the 3 cases where the although the determinant is <FLT\_MAX, operations on the coefficients might trigger an Floating Point Overflow error.\*/

if (a[1]\*a[1]>DBL\_MAX || fabs(a[0]\*a[2])>DBL\_MAX){

if (a[0]\*a[2]>0){//printf("DBL MAX1 \n");

/\*puting the denominator in the square root and factorising allows to divide the coefficients before multiplying them together\*/

dr= sqrt(((a[1]/(2\*a[2]))+(sqrt(a[0]/a[2])))\*((a[1]/(2\*a[2]))-(sqrt(a[0]/a[2]))));

if (a[1]>0){

root[1]=(-a[1])/(2\*a[2])-dr;

root[2]=(a[0]/a[2])/(root[1]);

if (root[1]<root[2]){

return(2);

}

else {

changeorder=root[1];

root[1]=root[2];

root[2]=changeorder;

}}

else {

root[1]=(-a[1]+dr)/(2\*a[2]);

root[2]=(a[0]/a[2])/(root[1]);

if (root[1]<root[2]){

return(2);

}

else {

changeorder=root[1];

root[1]=root[2];

root[2]=changeorder;

}

}

return(2);

}

else{

//printf("DBL MAX2 \n");

//I do not expand the dr above to force a[1] to be divided before it is multiplied.

dr=sqrt((a[1]/(2\*a[2]))\*(a[1]/(2\*a[2]))-(a[0]/a[2]));

if (a[1]>0){

root[1]=(-a[1])/(2\*a[2])+dr;

root[2]=(a[0]/a[2])/(root[1]);

if (root[1]<root[2]){

return(2);

}

else {

//ordering the roots

changeorder=root[1];

root[1]=root[2];

root[2]=changeorder;

}}

else {

root[1]=(-a[1]+dr)/(2\*a[2]);

root[2]=(a[0]/a[2])/(root[1]);

if (root[1]<root[2]){

return(2);

}

else {

//ordering the roots

changeorder=root[1];

root[1]=root[2];

root[2]=changeorder;

}

}

return(2);}

}

else

//case where operations on a[0],a[1],a[2] don't trigger floating point overflow.

{dr= sqrt(d);

//printf("No DBL MAX\n");

if (a[1]>0){

//computing the roots

root[1]=(-a[1]-dr)/(2\*a[2]);

root[2]=(a[0]/a[2])/(root[1]);

if (root[1]<root[2]){

return(2);

}

else {

changeorder=root[1];

root[1]=root[2];

root[2]=changeorder;

}}

else {

//printf("Normal distinct case \n");

root[1]=(-a[1]+dr)/(2\*a[2]);

root[2]=(a[0]/a[2])/(root[1]);

if (root[1]<root[2]){

return(2);

}

else {

changeorder=root[1];

root[1]=root[2];

root[2]=changeorder;

}

}

return(2);}

}

//case of repeated roots.

else if (d==0)

{root[1]=-a[1]/(2\*a[2]);

root[2]=root[1];

return(1);}

else {

//printf("Imaginary Roots\n");

dr=sqrt(-d);

root[1]=(-a[1]/(2\*a[2]));

root[2]=dr/(2\*a[2]);

return(0);}

}

Rcubic\_Roots:

#include <stdio.h>

#include <math.h>

#include <float.h>

int lin\_root(double \*,double \*);

int quad\_roots(double \*, double \*);

int rcubic\_roots(double \*,double \*);

int rcubic\_roots(double \*a,double \*root)

/\* <Elkrief>\_<Alexandre>\_exer\_1c \*/

{double p,alpha,beta,i,fx,df,ytemp,y,big,small,middle,delta\_quad,p2,y2;

double root\_inter1[3],root\_inter2[3];

int quad\_case;

if (a[1]==0 && a[2]==0){

//printf("a[2]=a[1]=%f\n",a[0]);

if (a[0]>0){

root[1]=-pow(fabs(a[0]),1.0/3.0);

//real part of x=-sign(a[0])|a[0]|^(1/3)\*cubic root of unity

root[2]=-pow(fabs(a[0]),1.0/3.0)\*(-0.5);

//imaginary part of x=-sign(a[0])|a[0]|^(1/3)\*cubic root of unity

root[3]=pow(fabs(a[0]),1.0/3.0)\*sqrt(3)\*0.5;

}

else{root[1]=pow(fabs(a[0]),1.0/3.0);

root[2]=pow(fabs(a[0]),1.0/3.0)\*(-0.5);

root[3]=pow(fabs(a[0]),1.0/3.0)\*sqrt(3)\*0.5;

}

return(0);

}

else if(a[0]==0){

//printf("a[0]=%f\n",a[0]);

root[1]=0;

double quad1[3]={a[1],a[2],1};

quad\_case=quad\_roots(quad1,root\_inter1);

//printf("root\_inter=%lf, %lf\n",root\_inter1[1],root\_inter1[2]);

root[2]=root\_inter1[1];

root[3]=root\_inter1[2];

//printf("quad\_case=%d\n",quad\_case);

if (root[2]<0 && root[3]>0){

middle=root[1];

small=root[2];

big=root[3];

}

else if(root[3]<0){

big=root[1];

middle=root[3];

small=root[2];

}

else{

big=root[3];

middle=root[2];

small=root[1];

}

root[1]=small;

root[2]=middle;

root[3]=big;

if (root[1]==root[2] && root[2]==root[3]){

return(1);

}

else if (root[1]==root[2] || root[2]==root[3] || root[1]==root[3]){

return(2);

}

else if (root[1]!=root[2] && root[2]!=root[3] && root[1]!=root[3]){

return(3);

}

else if(quad\_case==0){return(0);}

else{//printf("test");

return(0);}

}

else if(a[0]==a[1]\*a[2]){

// printf("a[0]=a[1]\*a[2]\n");

//printf("a[1]=%f\n",a[1]);

//printf("sqrt(a[1])=%f\n",sqrt(a[1]));

if(a[1]<0){

if (fabs(a[2])>sqrt(fabs(a[1]))){

root[1]=-a[2];

root[2]=-sqrt(-a[1]);

root[3]=sqrt(-a[1]);

}

else{

root[1]=-sqrt(-a[1]);

root[2]=-a[2];

root[3]=sqrt(-a[1]);

}

return(3);

}

else{

root[1]=-a[2];

root[2]=0;

root[3]=sqrt(a[1]);

return(0);

}

}

else if(3\*a[1]==a[2]\*a[2] && 27\*a[0]==a[2]\*a[2]\*a[2]){

//printf("3-27 case\n");

root[1]=-a[2]/3.0;

root[2]=-a[2]/3.0;

root[3]=-a[2]/3.0;

return(1);

}

else {

beta= -a[2]/3.0;

//printf("a[0] print=%f\n",a[0]);

//printf("beta print=%lf \n", beta);

if ((beta\*beta\*beta+a[2]\*beta\*beta+a[1]\*beta+a[0])>0){

//printf("neg\n");

alpha=-pow((beta\*beta\*beta+a[2]\*beta\*beta+a[1]\*beta+a[0]),1.0/3.0);

//alpha=-pow(fabs(((beta+a[2])\*beta+a[1])\*1+a[0]/beta),1.0/3.0)\*pow(fabs(beta),1.0/3.0);

}

else{

// printf("pos\n");

//printf("kk=%lf\n",beta\*beta\*beta+a[2]\*beta\*beta+a[1]\*beta+a[0]);

alpha=pow(fabs(beta\*beta\*beta+a[2]\*beta\*beta+a[1]\*beta+a[0]),1.0/3.0);

//alpha=pow(fabs(((beta+a[2])\*beta+a[1])\*1+a[0]/beta),1.0/3.0)\*pow(fabs(beta),1.0/3.0);

}

//printf("alpha=%lf, beta=%e \n",alpha,beta);

//p=(3\*beta\*beta+2\*a[2]\*beta+a[1])/(alpha\*alpha);

p=((3\*beta\*beta+2\*a[2]\*beta+a[1])/alpha)/alpha;

//printf("p=%lf \n",p);

//printf("Reduced cubic: y^3+%lf\*y-1=0 \n",p);

i=0;

p2=-3/pow(4,1.0/3.0);

y2=-1/pow(2,1.0/3.0);

//printf("p2=%f, y2=%f\n",p2,y2);

//special case Q3-vi

if (fabs(p-p2)<0.01){

//printf("p-p2 case\n");

if (alpha\*y2+beta<alpha\*(1/(y2\*y2))+beta){

root[1]=alpha\*y2+beta;

root[2]=alpha\*y2+beta;

root[3]=alpha\*(1/(y2\*y2))+beta;

}

else{

root[1]=alpha\*(1/(y2\*y2))+beta;

root[2]=alpha\*y2+beta;

root[3]=alpha\*y2+beta;

}

//printf("Roots=%lf %lf %lf\n", root[1],root[2],root[3]);

return(2);

}

//special case Q3-v

else if (p==0){

//printf("p=0 case\n");

root[1]=1;

root[2]=-0.5;

root[3]=sqrt(3)/2.0;

return(0);

}

else if (fabs(p)<2) {

y=1-p/3;

// printf("|p|<2\n");

}

else {

y=1/p;

//printf("|p|>2\n");

}

//printf("y=%f \n",y);

while (ytemp!=y && i<4){

fx=y\*y\*y+p\*y-1;

df=3\*y\*y+p;

ytemp=y;

y=y-fx/df;

//printf("%.25lf \n",y);

i=i+1;

}

root[1]=alpha\*y+beta;

double quad2[3]={(-a[0]/(root[1])),(a[2]+root[1]),1};

quad\_case=quad\_roots(quad2, root\_inter2);

//printf("root\_inter=%lf, %lf\n",root\_inter2[1],root\_inter2[2]);

root[2]=root\_inter2[1];

root[3]=root\_inter2[2];

//printf("Quadratic coeffs: %f, %f \n",a[2]+(root[1]),-a[0]/(root[1]));

//ordering the roots in ascending order

delta\_quad=(a[2]+(root[1]))\*(a[2]+(root[1]))-4\*(-a[0]/(root[1]));

//printf("Delta\_quad= %lf\n",delta\_quad);

if (delta\_quad>=0){

if (root[1]>root[2] && root[1]>root[3]){

big=root[1];

if (root[2]>root[3]){

middle=root[2];

small=root[3];

}

else{

middle=root[3];

small=root[2];

}

}

else if (root[2]>root[3]){

big=root[2];

if (root[1]>root[3]){

middle=root[1];

small=root[3];

}

else{

middle=root[3];

small=root[1];

}

}

else{

big=root[3];

if (root[1]>root[2]){

middle=root[1];

small=root[2];

}

else{

middle=root[2];

small=root[1];

}

}

root[1]=small;

root[2]=middle;

root[3]=big;

if (root[1]==root[2] && root[2]==root[3]){

return(1);

}

else if (root[1]==root[2] || root[2]==root[3] || root[1]==root[3]){

return(2);

}

else {

return(3);

}

}

else{

return(0);

}

}

}

Using the following test problems from Exercise1b we get the right results :

Test 1 :

Output 1: Roots= -2.000000, 1.000000, 3.000000

Test 2 :

Output 2: Roots= 1.000000, -2.000000, 3.000000

Test 3 :

Output 3 : Roots= -1.000000, -1.000000, 1.000000

Question 2 :

Below are the codes for the rquartic\_roots and its Main program function. It is already improved to treat the special cases (i), (ii), (iii).

Main():

#include <stdio.h>

#include <math.h>

#include <float.h>

int lin\_root(double \*,double \*);

int quad\_roots(double \*, double \*);

int rcubic\_roots(double \*,double \*);

int rquartic\_roots(double \*, double \*);

int main(void){

printf(" Name: <Alexandre Elkrief> \n CID: <00732974> \n LIBRARY NO: <0246431227> \n Email Address: <ae2212@ic.ac.uk> \n Course Code: <M3SC>\n");

double A,B,C;

double a[4],root[5];

int rcubic\_case, quad\_case,rquartic\_case;

scanf("%lf %lf %lf %lf",&a[3] , &a[2], &a[1], &a[0]);

rquartic\_case=rquartic\_roots(a,root);

printf("Roots= %lf, %lf, %lf %lf \n",root[1], root[2], root[3],root[4]);

printf("RQuartic\_Case=%d\n",rquartic\_case);

}

Rquartic\_roots:

#include <stdio.h>

#include <math.h>

#include <float.h>

int lin\_root(double \*,double \*);

int quad\_roots(double \*, double \*);

int rcubic\_roots(double \*,double \*);

int rquartic\_roots(double \*a, double \*root)

/\* <Elkrief>\_<Alexandre>\_exer\_1c \*/

{double b[3]={(4\*a[0]\*a[2]-a[1]\*a[1]-a[0]\*a[3]\*a[3]),(a[1]\*a[3]-4\*a[0]),-a[2]};

double root\_inter1[4],r,quad\_roots1[3],quad\_roots2[3],r1,r2,r3,r4;

int rcubic\_case,quad\_case1, quad\_case2,quad\_case\_spec,cub\_inter\_case;

if (a[0]==0){

//printf("a0=0\n");

double cub\_inter\_coeffs[3]={a[1],a[2],a[3]},cub\_inter\_roots[4];

cub\_inter\_case=rcubic\_roots(cub\_inter\_coeffs,cub\_inter\_roots);

//printf("cub\_inter\_case=%d\n",cub\_inter\_case);

//printf("cub\_inter\_roots= %lf %lf %lf \n", cub\_inter\_roots[1],cub\_inter\_roots[2],cub\_inter\_roots[3]);

if (cub\_inter\_case==0){

if (cub\_inter\_roots[1]<=0){

root[1]=cub\_inter\_roots[1];

root[2]=0;

root[3]=cub\_inter\_roots[2];

root[4]=cub\_inter\_roots[3];

}

else{

root[1]=0;

root[2]=cub\_inter\_roots[1];

root[3]=cub\_inter\_roots[2];

root[4]=cub\_inter\_roots[3];

}

return(2);

}

else{//dont forget cases of repeated roots for appropriate return value!!!!!

if (cub\_inter\_roots[1]>=0){

root[1]=0;

root[2]=cub\_inter\_roots[1];

root[3]=cub\_inter\_roots[2];

root[4]=cub\_inter\_roots[3];

}

else if(cub\_inter\_roots[1]<0 && cub\_inter\_roots[2]>=0){

root[1]=cub\_inter\_roots[1];

root[2]=0;

root[3]=cub\_inter\_roots[2];

root[4]=cub\_inter\_roots[3];

}

else if(cub\_inter\_roots[2]<0 && cub\_inter\_roots[3]>=0){

root[1]=cub\_inter\_roots[1];

root[2]=cub\_inter\_roots[2];

root[3]=0;

root[4]=cub\_inter\_roots[3];

}

else{

root[1]=cub\_inter\_roots[1];

root[2]=cub\_inter\_roots[2];

root[3]=0;

root[4]=cub\_inter\_roots[3];

}

if (cub\_inter\_case==3){//3 distinct roots

//printf("cub case 3 ordering\n");

if(cub\_inter\_roots[1]!=0 && cub\_inter\_roots[2]!=0 && cub\_inter\_roots[3]!=0){

return(4);

}

else{return(3);}

}

else if (cub\_inter\_case==2){//1 repeated, 1 real root

if(cub\_inter\_roots[1]!=cub\_inter\_roots[2]){//repeated roots are r2,r3

if(cub\_inter\_roots[1]==0){return(2);}

else if(cub\_inter\_roots[2]==0){return(1);}

else{return(3);}

}

else{

if(cub\_inter\_roots[1]==0){return(1);}

else if(cub\_inter\_roots[3]==0){return(2);}

else{return(3);}

}

}

else{//triple/quadruple root

return(1);

}

}

}

else if (a[1]==0 && a[2]==0 && a[3]==0){

if (a[0]>0){

root[1]=0;

root[2]=0;

root[3]=-pow(fabs(a[0]),1.0/4.0);

root[4]=pow(fabs(a[0]),1.0/4.0);

return(0);

}

else{

root[1]=-pow(fabs(a[0]),1.0/4.0);

root[2]=-pow(fabs(a[0]),1.0/4.0);

root[3]=pow(fabs(a[0]),1.0/4.0);

root[4]=pow(fabs(a[0]),1.0/4.0);

return(2);

}

}

//Q2-(ii)

else if(a[1]==0 && a[3]==0){

//printf("a1=a3=0\n");

double quad\_special[3]={a[0],a[2],1},quad\_inter1, quad\_inter2;

quad\_case\_spec=quad\_roots(quad\_special,root);

quad\_inter1=root[1];

quad\_inter2=root[2];

if (quad\_case\_spec==0){//imaginary roots of quadratric

root[1]=(1/sqrt(2))\*sqrt(sqrt(quad\_inter1\*quad\_inter1+quad\_inter2\*quad\_inter2)+quad\_inter1);

if(quad\_inter2>=0){

root[2]=(1/sqrt(2))\*sqrt(sqrt(quad\_inter1\*quad\_inter1+quad\_inter2\*quad\_inter2)-quad\_inter1);

}

else{root[2]=-(1/sqrt(2))\*sqrt(sqrt(quad\_inter1\*quad\_inter1+quad\_inter2\*quad\_inter2)-quad\_inter1);}

root[3]=-root[1];

root[4]=-root[2];

if(root[3]<=root[1]){

r1=root[3];

r2=root[4];

r3=root[1];

r4=root[2];}

else{

r1=root[1];

r2=root[2];

r3=root[3];

r4=root[4];}

root[1]=r1;

root[2]=r2;

root[3]=r3;

root[4]=r4;

return(0);

}

else{

if(quad\_inter1>0 && quad\_inter2>0){

root[1]=-sqrt(quad\_inter2);

root[2]=-sqrt(quad\_inter1);

root[3]=sqrt(quad\_inter1);

root[4]=sqrt(quad\_inter2);

return(4);

}

else if(quad\_inter1<0 && quad\_inter2>0){

//printf("r3, r4 are complex on imaginary conjugates, r1,r2 are real\n");

root[1]=-sqrt(quad\_inter2);

root[2]=sqrt(quad\_inter2);

root[3]=-sqrt(fabs(quad\_inter1));

root[4]=sqrt(fabs(quad\_inter1));

return(2);

}

else {

// printf("all imaginary parts\n");

root[1]=-sqrt(fabs(quad\_inter1));

root[2]=-sqrt(fabs(quad\_inter2));

root[3]=sqrt(fabs(quad\_inter2));

root[4]=sqrt(fabs(quad\_inter1));

return(2);

}

}

}

else{

//printf("No special case\n");

rcubic\_case=rcubic\_roots(b,root\_inter1);

//printf("cubic case=%d\n",rcubic\_case);

//printf("Cubic roots= %lf %lf %lf \n", root\_inter1[1],root\_inter1[2],root\_inter1[3]);

if (rcubic\_case==0){

r=root\_inter1[1];

}

else{

r=root\_inter1[3];

}

//printf("r=%lf \n \n",r);

double quad\_coeff1[3]={((r/2)+sqrt((r/2)\*(r/2)-a[0])),((a[3]/2)+sqrt(((a[3]\*a[3])/4)+r-a[2])),1};

double quad\_coeff2[3]={((r/2)-sqrt((r/2)\*(r/2)-a[0])),((a[3]/2)-sqrt(((a[3]\*a[3])/4)+r-a[2])),1};

double quad\_coeff3[3]={((r/2)+sqrt((r/2)\*(r/2)-a[0])),((a[3]/2)-sqrt(((a[3]\*a[3])/4)+r-a[2])),1};

double quad\_coeff4[3]={((r/2)-sqrt((r/2)\*(r/2)-a[0])),((a[3]/2)+sqrt(((a[3]\*a[3])/4)+r-a[2])),1};

if (r>=0){

quad\_case1=quad\_roots(quad\_coeff1,quad\_roots1);

quad\_case2=quad\_roots(quad\_coeff2,quad\_roots2);

}

else{

quad\_case1=quad\_roots(quad\_coeff3,quad\_roots1);

quad\_case2=quad\_roots(quad\_coeff4,quad\_roots2);

}

if (quad\_case1==0 && quad\_case2==0){

//4 imaginary

if (quad\_roots1[1]<quad\_roots2[1]){

root[1]=quad\_roots1[1];

root[2]=quad\_roots1[2];

root[3]=quad\_roots2[1];

root[4]=quad\_roots2[2];

}

else {

root[1]=quad\_roots2[1];

root[2]=quad\_roots2[2];

root[3]=quad\_roots1[1];

root[4]=quad\_roots1[2];

}

return(0);

}

else if (quad\_case1==0 && quad\_case2!=0){

//2 real, 2 imaginary

root[1]=quad\_roots2[1];

root[2]=quad\_roots2[2];

root[3]=quad\_roots1[1];

root[4]=quad\_roots1[2];

return(2);

}

else if (quad\_case1!=0 && quad\_case2==0){

//2 real, 2 imaginary

root[1]=quad\_roots1[1];

root[2]=quad\_roots1[2];

root[3]=quad\_roots2[1];

root[4]=quad\_roots2[2];

return(2);

}

else if ((quad\_case1==1) && (quad\_case2==1)){

//2 equal, 2 equal

if (quad\_roots1[1]<quad\_roots2[1]){

root[1]=quad\_roots1[1];

root[2]=quad\_roots1[2];

root[3]=quad\_roots2[1];

root[4]=quad\_roots2[2];

}

else{

root[1]=quad\_roots2[1];

root[2]=quad\_roots2[2];

root[3]=quad\_roots1[1];

root[4]=quad\_roots1[2];

}

return(1);

}

else if (quad\_case1==1 || quad\_case2==1){

if (quad\_case1==2){//2 real 1 repeated

if(quad\_roots1[2]<quad\_roots2[1]){

root[1]=quad\_roots1[1];

root[2]=quad\_roots1[2];

root[3]=quad\_roots2[1];

root[4]=quad\_roots2[2];

}

else if (quad\_roots1[1]>quad\_roots2[1]){

root[1]=quad\_roots2[1];

root[2]=quad\_roots2[2];

root[3]=quad\_roots1[1];

root[4]=quad\_roots1[2];

}

else {

root[1]=quad\_roots1[1];

root[2]=quad\_roots2[1];

root[3]=quad\_roots2[2];

root[4]=quad\_roots1[2];

}

}

else if (quad\_case2==2){//1 repeated, 2 real

if(quad\_roots2[2]<quad\_roots1[1]){

root[1]=quad\_roots2[1];

root[2]=quad\_roots2[2];

root[3]=quad\_roots1[1];

root[4]=quad\_roots1[2];

}

else if (quad\_roots1[1]<quad\_roots2[1]){

root[1]=quad\_roots1[1];

root[2]=quad\_roots1[2];

root[3]=quad\_roots2[1];

root[4]=quad\_roots2[2];

}

else {

root[1]=quad\_roots2[1];

root[2]=quad\_roots1[1];

root[3]=quad\_roots1[2];

root[4]=quad\_roots2[2];

}

}

if (root[2]==root[3]){return(1);

}

else{return(3);}

}

else {

if (quad\_roots1[1]<quad\_roots2[1]){

//start with r1

if(quad\_roots1[2]<quad\_roots2[1]){//r1 r2 r3 r4

root[1]=quad\_roots1[1];

root[2]=quad\_roots1[2];

root[3]=quad\_roots2[1];

root[4]=quad\_roots2[2];

}

else if ((quad\_roots1[2]>quad\_roots2[1]) && (quad\_roots1[2]<quad\_roots2[2])){//r1 r3 r2 r4

root[1]=quad\_roots1[1];

root[2]=quad\_roots2[1];

root[3]=quad\_roots1[2];

root[4]=quad\_roots2[2];

}

else{//r1 r3 r4 r2

root[1]=quad\_roots1[1];

root[2]=quad\_roots2[1];

root[3]=quad\_roots2[2];

root[4]=quad\_roots1[2];

}

}

else{//start with r3

if(quad\_roots2[2]<quad\_roots1[1]){//r3 r4 r1 r2

root[1]=quad\_roots2[1];

root[2]=quad\_roots2[2];

root[3]=quad\_roots1[1];

root[4]=quad\_roots1[2];

}

else if ((quad\_roots2[2]>quad\_roots1[1]) && (quad\_roots2[2]<quad\_roots1[2])){//r3 r1 r4 r2

root[1]=quad\_roots2[1];

root[2]=quad\_roots1[1];

root[3]=quad\_roots2[2];

root[4]=quad\_roots1[2];

}

else{//r3 r1 r2 r4

root[1]=quad\_roots2[1];

root[2]=quad\_roots1[1];

root[3]=quad\_roots1[2];

root[4]=quad\_roots2[2];

}

}

return(4);

}

}

}

Question 3:

The code which outputs the required table is the following:

#include <stdio.h>

#include <math.h>

#include <float.h>

#define PI 3.14159265

int lin\_root(double \*,double \*);

int quad\_roots(double \*, double \*);

int rcubic\_roots(double \*,double \*);

int rquartic\_roots(double \*, double \*);

int main(void){

printf(" Name: <Alexandre Elkrief> \n CID: <00732974> \n LIBRARY NO: <0246431227> \n Email Address: <ae2212@ic.ac.uk> \n Course Code: <M3SC>\n\n");

double root[5],a[4],b,X,Y,theta1, theta2, theta3, theta4,theta\_max1, theta\_max2,theta\_max3,theta\_max4;

double x1,x2,x3,x4,y1,y2,y3,y4;

int rcubic\_case, quad\_case,rquartic\_case;

X=0.2;

Y=1.0/3.0;

printf(" B Root1 Root2 Root3 Root4 Quad\_Case Phi1 Phi2 Phi3 Phi4\n");

for(b=0.05;b<0.96;b=b+0.05){

a[0]=-1;

a[1]=(2\*X-2+2\*b\*b)/(b\*Y);

a[2]=0;

a[3]=(2\*X+2-2\*b\*b)/(b\*Y);

rquartic\_case=rquartic\_roots(a,root);

theta1=2\*atan(root[1])\*(180/PI);

theta2=2\*atan(root[2])\*(180/PI);

theta3=2\*atan(root[3])\*(180/PI);

theta4=2\*atan(root[4])\*(180/PI);

if (rquartic\_case==4){

printf("%.2f %11lf %10lf %10lf %10lf %8d %10.2f %10.2f %10.2f %10.2f\n",b,root[1], root[2], root[3],root[4],rquartic\_case,theta1, theta2, theta3, theta4);

}

else{

printf("%.2f %11lf %10lf %10lf I\*%8lf %8d %10.2f %10.2f %10.2f %10.2f\n",b,root[1], root[2], root[3],root[4],rquartic\_case,theta1, theta2, theta3, theta4);

}

}

b=0.64429054104497;

a[0]=-1;

a[1]=(2\*X-2+2\*b\*b)/(b\*Y);

a[2]=0;

a[3]=(2\*X+2-2\*b\*b)/(b\*Y);

rquartic\_case=rquartic\_roots(a,root);

theta\_max1=2\*atan(root[1]);

theta\_max2=2\*atan(root[2]);

theta\_max3=2\*atan(root[3]);

theta\_max4=2\*atan(root[4]);

x1=cos(theta\_max1);

x2=cos(theta\_max2);

x3=cos(theta\_max3);

x4=cos(theta\_max4);

y1=b\*sin(theta\_max1);

y2=b\*sin(theta\_max2);

y3=b\*sin(theta\_max3);

y4=b\*sin(theta\_max4);

//the following are all the values for the maximum b for which there are 4 real roots.

//The angles are computed in radians to be able to reuse them in R to plot the elipse and the normal lines.

printf("B\_MAX: \n");

printf("%lf %11lf %10lf %10lf %10lf %8d %10.2f %10.2f %10.2f %10.2f\n",b,root[1], root[2], root[3],root[4],rquartic\_case, theta\_max1,theta\_max2, theta\_max3, theta\_max4);

printf("t1=%lf, phi1=%lf, x1=%lf, y1=%lf\n",root[1],theta\_max1,x1,y1);

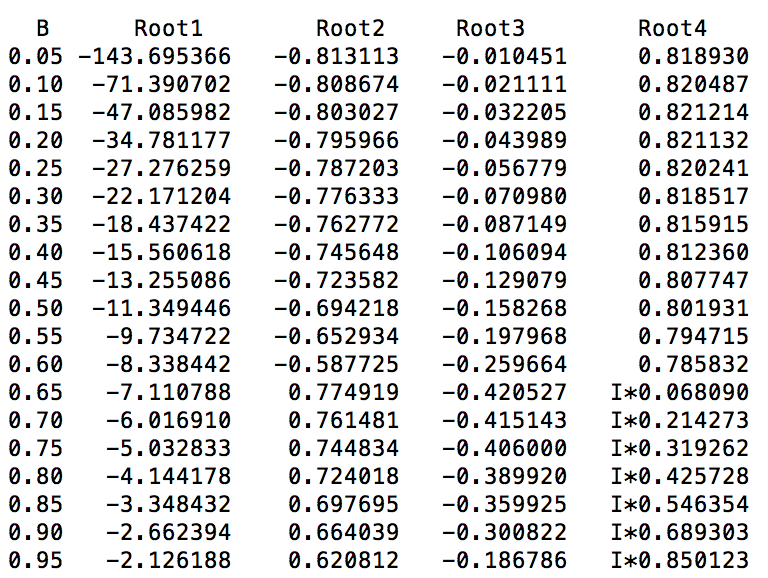
printf("t2=%lf, phi2=%lf, x2=%lf, y2=%lf\n",root[2],theta\_max2,x2,y2);

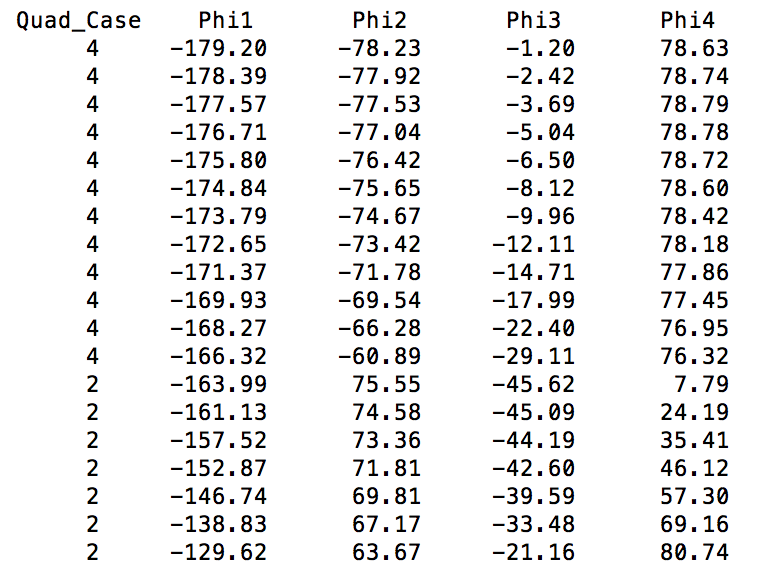
printf("t3=%lf, phi3=%lf, x3=%lf, y3=%lf\n",root[3],theta\_max3,x3,y3);

printf("t4= %lf, phi4=%lf, x4=%lf, y4=%lf\n",root[4],theta\_max4,x4,y4);

}

The table is the following:





The maximum value of b for which there are 4 real roots is 0.64429054104497

The R-code for the plot is below:

i=0

k=1

x=rep(0,360)

y=rep(0,360)

X=1/5

Y=1/3

while (i<(2\*pi)){

x[k]=cos(i)

y[k]=0.64429054104497\*sin(i)

k=k+1

i=i+(2\*pi)/360

}

phi1<--2.867215

phi2<--0.798204

phi3<--0.797394

phi4<-1.321220

x1<-cos(phi1)

x2<-cos(phi2)

x3<-cos(phi3)

x4<-cos(phi4)

y1<-0.64429054104497\*sin(phi1)

y2<-0.64429054104497\*sin(phi2)

y3<-0.64429054104497\*sin(phi3)

y4<-0.64429054104497\*sin(phi4)

m1=(y1-Y)/(x1-X)

m2=(y2-Y)/(x2-X)

m3=(y3-Y)/(x3-X)

m4=(y4-Y)/(x4-X)

xvals1<-c(X,x1,-1.5,1.5)

xvals2<-c(X,x2, -1.5,1.5)

xvals3<-c(0,0.5)

xvals4<-c(X,x4,-1.5,1.5)

l1<-c(Y,y1,Y+m1\*(-1.5-X),Y+m1\*(1.5-X))

l2<-c(Y,y2,Y+m2\*(-1.5-X),Y+m2\*(1.5-X))

l3<-c(Y+m3\*(0-X),Y+m3\*(0.5-X))

l4<-c(Y,y4,Y+m4\*(-1.5-X),Y+m4\*(1.5-X))

plot(x,y,type='l')

abline(h=0,v=0)

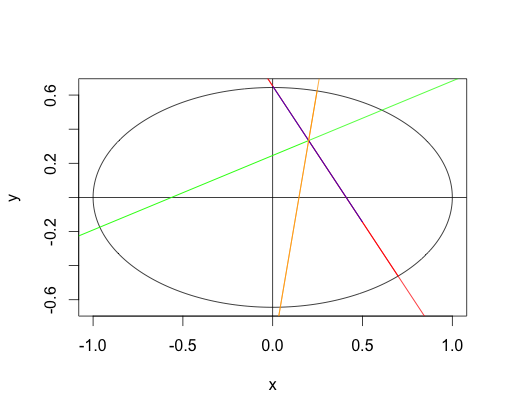
lines(xvals1,l1,col='green')

lines(xvals2,l2,col='red')

lines(xvals3,l3,col='blue')

lines(xvals4,l4,col='orange')

The plot is the following:



We can see that the shortest to distance from the the point (X,Y) to the ellipse is the short part of the orange line.

Question 4:

If B=1-A :

If x0=x1=c :

The code is the following:

#include <stdio.h>

#include <math.h>

#include <float.h>

int lin\_root(long double \*,long double \*);

int quad\_roots(long double \*, long double \*);

int rcubic\_roots(long double \*,long double \*);

int rquartic\_roots(long double \*, long double \*);

int main(void){

printf(" Name: <Alexandre Elkrief> \n CID: <00732974> \n LIBRARY NO: <0246431227> \n Email Address: <ae2212@ic.ac.uk> \n Course Code: <M3SC>\n");

long double A,B,x[26] ;

int I1,I2,I3,i;

scanf("%d %d %d",&I1,&I2,&I3);

A=I1+1;

B=-I1;

x[0]=(long double)I2/(long double)I3;

x[1]=x[0];

//printf("A,B= %lf, %lf\n",A,B);

printf("x[0]=%Lf\nx[1]=%Lf\n",x[0],x[1]);

for (i=2;i<26;i=i+1){

x[i]=A\*x[i-1]+B\*x[i-2];

printf("x[%d]=%Le\n",i,x[i]);

}

return(0);

}

The results for cases 1,4 and 7 are the same. Namely:

x[0]=x[1]=……=x[25]=0.5

This agrees with the exact solution.

For cases 3,6 and 9 the solutions are the following:

x[0]=x[1]=……=x[25]=0.25

These also agree with the exact solutions.

For cases 2,5 and 8, the exact solutions are:

x[0]=x[1]=……=x[25]=1/3

However due to the fact that the compiler computes 1002\*0.33-1001\*0.33 there are approximations when computing the 0.33 and then multiplying by 0.33. These errors accumulate at every step of the algorithm yielding big errors. Moreover the bigger the I1 the larger the error is. Therefore the error for case 8 is the biggest.

The results are the following:

Case 2:

x[0]=0.333333

x[1]=0.333333

x[2]=0.333333

x[3]=0.333333

x[4]=0.333348

x[5]=0.484992

x[6]=1517.067543

x[7]=15168859.166507

x[8]=151703757190.899292

x[9]=1517189275662850.500000

x[10]=15173409945904164864.000000

x[11]=151749272868987565244416.000000

x[12]=1517644477962744587630936064.000000

x[13]=15177962424105408860865404338176.000000

x[14]=151794802203478200070352807972044800.000000

x[15]=1518099816836985461250064353988379148288.000000

x[16]=15182516268186689899723348500587446693527552.000000

x[17]=151840345198135089700534815717577905604929781760.000000

x[18]=1518555292326548958814702793998011392233483851005952.000000

x[19]=15187071478557816326853139041070852790745529848929714176.000000

x[20]=151885901857056724065731777777170538699407605121435603828736.000000

x[21]=1519010904472424306680794257336752407553169978513844221885022208.000000

x[22]=15191628055628716392130382782126509900510952324705437680954519846912.000000

x[23]=151931472184342796681448840449781302941039006230761322616203797657026560.000000

x[24]=1519466653315612235193186388890699848831999462894830030958682407132200960000.000000

x[25]=15196185999809438662582504561359537695594101012188991260830258926575477843820544.000000

Case 5:

x[0]=0.333333

x[1]=0.333333

x[2]=0.333333

x[3]=0.333333

x[4]=0.333333

x[5]=0.333371

x[6]=0.371457

x[7]=38.494974

x[8]=38200.135886

x[9]=38238002.688755

x[10]=38276240358.110352

x[11]=38314516598135.125000

x[12]=38352831114732920.000000

x[13]=38391183945847652352.000000

x[14]=38429575129793497661440.000000

x[15]=38468004704923290697728000.000000

x[16]=38506472709628213546044096512.000000

x[17]=38544979182337840044352001277952.000000

x[18]=38583524161520175353373362697011200.000000

x[19]=38622107685681696298878301137081991168.000000

x[20]=38660729793367374637574610117901326221312.000000

x[21]=38699390523160740455115729064376850525978624.000000

x[22]=38738089913683899457503029636767321418134192128.000000

x[23]=38776828003597581496049451531363327342392020828160.000000

x[24]=38815604831601179565621405685168487751601067795677184.000000

x[25]=38854420436432782382795917897870011656842967128857378816.000000

Case 8:

x[0]=0.333333

x[1]=0.333333

x[2]=0.333333

x[3]=0.333334

x[4]=0.357587

x[5]=2425.773806

x[6]=242546473.060891

x[7]=24254889819228.843750

x[8]=2425513236812670464.000000

x[9]=242553749194503851343872.000000

x[10]=24255617473199580881228922880.000000

x[11]=2425586002937431214634628781441024.000000

x[12]=242561025879746036459689724252096823296.000000

x[13]=24256345149000484031376074963620939199152128.000000

x[14]=2425658771245197331276422976599434275749592301568.000000

x[15]=242568302783291016917369012250343229515715258718617600.000000

x[16]=24257072846631885780418221952591448960884043233114137296896.000000

x[17]=2425731541736035375948287837092137792592711852869455120286351360.000000

x[18]=242575579905145264983961045547429207304064307649268037199258498105344.000000

x[19]=24257800566094431119930737424870950815648638300616289486258954062786461696.000000

x[20]=2425804314410009546806312376302671052275196741558848644481707996739444452360192.000000

x[21]=242582857245315350325746010807175262906768470218235190050022542333605767578789085184.000000

x[22]=24258528307388778981055630802190895876322088878398462086789082033095885571693447672233984.000000

x[23]=2425877089267185214991537070017433407783709904699370198120675228522686247086760130074371948544.000000

x[24]=242590134803807781237547855508534198131081586301104989322766389710821192889326957050658075822784512.000000

x[25]=24259256070515581720179718464837810834615457567756766491155355169971088133932306766210506085918855659520.000000

Replacing type double by float yields a smaller range of possible values. The algorithm therefore yields a “-inf” and then “nan” for the following values as “inf” is not usable in the algorithm.

Replacing double by long double slightly reduces the error as 0.33 is computed more accurately.